CodeReviewQA: The Code Review Comprehension Assessment for Large Language Models

Hong Yi Lin♠, Chunhua Liu♠, Haoyu Gao♠
Patanamon Thongtanunam♠, Christoph Treude♣
♠The University of Melbourne, ♣Singapore Management University
{tom.lin1, chunhua.liu1, haoyu.gao3, patanamon.t}@unimelb.edu.au
ctreude@smu.edu.sg

Abstract

State-of-the-art large language models (LLMs) have demonstrated impressive code generation capabilities but struggle with real-world software engineering tasks, such as revising source code to address code reviews, hindering their practical use. Code review comments are often implicit, ambiguous, and colloquial, requiring models to grasp both code and human intent. This challenge calls for evaluating large language models' ability to bridge both technical and conversational contexts. While existing work has employed the automated code refinement (ACR) task to resolve these comments, current evaluation methods fall short, relying on text matching metrics that provide limited insight into model failures and remain susceptible to training data contamination. To address these limitations, we introduce a novel evaluation benchmark, **CodeReviewQA** that enables us to conduct fine-grained assessment of model capabilities and mitigate data contamination risks. In CodeReviewQA, we decompose the generation task of code refinement into **three essential reasoning steps**: *change type recognition* (CTR), *change localisation* (CL), and *solution identification* (SI). Each step is reformulated as multiple-choice questions with varied difficulty levels, enabling precise assessment of model capabilities, while mitigating data contamination risks. Our comprehensive evaluation spans 72 recently released large language models on **900 manually curated, high-quality examples** across nine programming languages. Our results show that CodeReviewQA is able to expose specific model weaknesses in code review comprehension, disentangled from their generative automated code refinement results. [1]

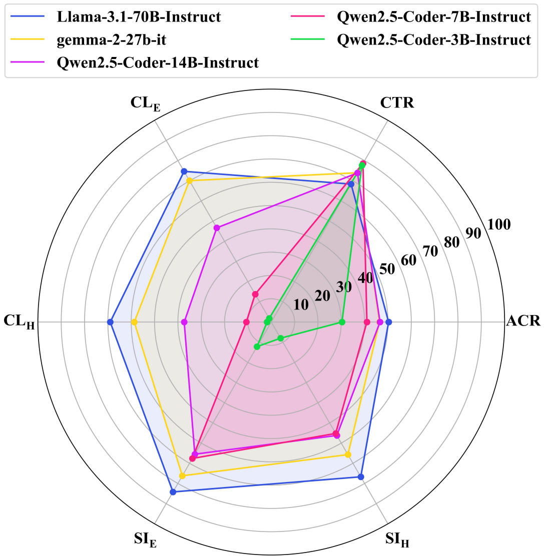

Figure 1: **CodeReviewQA** results (%) of the top performing model (per scale class) based on ACR Accuracy. **ACR:** Automated Code Refinement, **CTR:** Change Type Recognition, **CL:** Change Localisation, **SI:** Solution Identification, **E:** Easy, **H:** Hard.

1 Introduction

The proficiency of state-of-the-art large language models (LLMs) in code generation has garnered significant attention (Zhuo et al., 2025), demonstrating their ability to follow explicit instructions to author code. However, their competency in real-world software engineering environments remains limited (Pornprasit and Tantithamthavorn, 2024), particularly in collaborative tasks involving colloquial and complex forms of communication. A quintessential example is code reviewing, where review comments (Yang et al., 2023; Efstathiou and Spinellis, 2018) represent natural communication between developers with a shared mental model, often resulting in under-specified, ambiguous, and implicit expressions of intent. For example, this comment *"For all of the fuzz tests, does it make*

[1]Data Availability: https://huggingface.co/datasets/Tomo-Melb/CodeReviewQA

Findings of the Association for Computational Linguistics: ACL 2025, pages 9138–9166
July 27 - August 1, 2025 ©2025 Association for Computational Linguistics

sense to have versions for 'len_prefixed' both 'true' and 'false'?" is a rhetorical question that expresses an intended code change without explicitly detailing the literal change requirement and instructions.

As a result, the ability to resolve code review comments not only requires proficiency in understanding and generating code, but also the ability to comprehend the communicative intent behind natural language code reviews in relation to the source code that they address. Therefore, assessing how LLMs resolve code review comments serves as a crucial testbed for their proficiency in understanding and following implicit, conversational instructions in software development. Success in this domain would significantly advance automated software development assistance, potentially reducing developer workload and improving code quality.

To evaluate model capability in resolving code review comments, prior work has explored the *automated code refinement* (ACR) task using both small scale neural language models (Tufano et al., 2022; Thongtanunam et al., 2022) and LLMs (Guo et al., 2024; Pornprasit and Tantithamthavorn, 2024), aiming to automatically revise source code based on code review comments. While these efforts have advanced this direction, several critical challenges remain unaddressed. Firstly, current automatic evaluation approaches rely heavily on metrics such as exact match and BLEU (Papineni et al., 2002), which merely capture surface-level token similarities between the generated output and the ground truth, without reflecting intermediate comprehension capabilities. Secondly, as these evaluation benchmarks typically use popular GitHub projects, they risk data contamination from training data in LLMs (Sallou et al., 2024), potentially obfuscating true model capabilities. As a result, there are currently no suitable approaches for assessing the capabilities of state-of-the-art LLMs in automated code refinement.

To address these challenges, we introduce a novel evaluation benchmark that enables comprehensive assessment of automated code refinement capabilities. Our benchmark decomposes the original one-step generative task into three underlying reasoning steps: *change type recognition* (CTR), *change localisation* (CL), and *solution identification* (SI). These components represent essential cognitive processes required for understanding code review comments, and subsequently generating the required code revision. By reflecting explicit intermediate reasoning steps, our benchmark provides fine-grained feedback of model failures to support future model development.

To mitigate potential data contamination, we formulate each reasoning step as a multiple-choice question answering (MCQA) probe with synthetic answers. This approach transforms the original task into unfamiliar formats with unseen solutions, demanding proficiency in code review comprehension, rather than sequence memorisation of contaminated data (Zhu et al., 2024). Furthermore, we leverage MCQA's flexibility to introduce distractor variation strategies, enabling assessment of model understanding across different difficulty levels.

To avoid the pervasive issue of noisy examples present in past benchmarks (Tufano et al., 2024) and ensure high-quality evaluation data, we manually verify and curate 900 valid code refinement examples that cannot be automated by traditional software engineering tools. These examples are sourced from 199 repositories, reflecting nine of the most popular programming languages on GitHub. Finally, we evaluate 72 state-of-the-art code-intelligent LLMs, providing an extensive benchmark to facilitate future research. Our contributions can be summarised as follows:

1) CodeReviewQA Benchmark. The first ACR evaluation approach to include intermediate reasoning probes for detailed feedback on code review comprehension capabilities of LLMs. It is also the first approach to counteract the effects of data contamination in ACR assessment, allowing for the reuse of existing code review data in LLM evaluation. The benchmark consists of four tasks (three MCQA probes and one generative task) in total.

2) Clean Code Review Evaluation Set. The first code review evaluation set that has been completely manually verified for examples that are noisy or do not faithfully represent the task of ACR. To achieve diversity, the examples represent 199 repositories across nine of the most popular programming languages on GitHub. The evaluation set consists of 900 real code reviews in total.

3) Comprehensive Evaluation of LLMs. A large scale evaluation of state-of-the-art open source LLMs that have been trained on both code and natural language. These models span across five different scales, ranging from 1B to 72B parameters. This includes code focused models e.g., CodeLlama-70b-Instruct-hf, general purpose models e.g., Qwen2.5-72B-Instruct, and the latest reasoning models e.g., QwQ-32B. In total, we include 72 LLMs developed by 18 different organisations.

2 Background and Related Work

Recently, LLMs have shown promise in various software engineering tasks involving natural language artifacts. However, these artifacts vary significantly in their linguistic nature and structure. Some tasks involve explicit, non-conversational language, such as bug reports (Saha et al., 2018) and GitHub issues (Jimenez et al., 2024), which typically contain detailed specifications of defects or feature requests. Other tasks involve static monologues, like commit messages (Jiang et al., 2017), code comments (Hu et al., 2018), and pull request descriptions (Liu et al., 2019), which aim to clearly explain source code or code changes.

In contrast, code reviews are unique as they represent routine conversations in highly collaborative scenarios. As such, they are informal, free-flowing, and can lean on the interlocutor's shared technical knowledge, without being overly specific (Yang et al., 2023). Thus, interpreting code reviews requires a deep understanding of conversational language in a highly technical context, posing challenges for computational models. The automated code refinement task involves revising source code to address code review comments, which requires an understanding of both technical implications and the reviewer's unstated expectations. Such nuanced communication makes automated code refinement an ideal testbed for evaluating LLMs' ability to bridge both technical and conversational understanding in software development.

The automated code refinement task was typically framed as a sequence-to-sequence neural machine translation problem, where models "translate" pre-review code submissions into post-review code revisions that reflect the intent of the accompanying code review comments. Formally, this problem requires the following estimation:

$$P(H_{post}|H_{pre}, R_{nl}) \quad (1)$$

where H_{pre} denotes the submitted pre-review code hunk, R_{nl} denotes the natural language code review comment, and H_{post} denotes the expected post-review revision of that code hunk. The "hunk" refers to the code snippet within the file, where the code review comment was inlined. See Figure 2 for a concrete example.

While prior work has applied various neural language models, such as recurrent neural networks (Tufano et al., 2019) and transformers (Tufano et al., 2022; Thongtanunam et al., 2022), the task remains a challenging problem, even for recent LLMs such as GPT-4 (Guo et al., 2024; Lu et al., 2023; Tufano et al., 2024).

Code Review Benchmark	Size	#Lang	Metric	DC	MV	VD
Tufano 2021 (Tufano et al., 2021)	1.7k	1	Text Match	✗	✗	✗
T5CR (Tufano et al., 2022)	16.8k	1	Text Match	✗	✗	✗
CodeReviewer (Li et al., 2022)	13.1k	9	Text Match	✗	✗	✗
CodeReview-New (Guo et al., 2024)	14.6k	16	Text Match	✗	✗	✗
CodeReviewQA (Ours)	900	9	Text Match & Probe	✓	✓	✓

DC: Addresses Data Contamination, **MV:** Manual Verification, **VD:** Varied Difficulty

Table 1: Benchmarks for ACR.

Indeed, prior work has highlighted several limitations in the evaluation (Guo et al., 2024). Traditional evaluation approaches have relied heavily on text matching metrics such as exact match and BLEU (Tufano et al., 2024; Guo et al., 2024), which are either too strict or fail to provide meaningful feedback. The emergence of LLMs has introduced additional challenges, as they are trained on extensive code repositories, creating significant risks of training data contamination. While some researchers have attempted to address this by collecting code reviews that outpace training cutoff dates (Guo et al., 2024), such approaches lack long-term sustainability as real code reviews take time to naturally occur. Furthermore, existing benchmarks have been constructed automatically through large-scale mining using general heuristics, as a result, significant proportions of noise have been reported (Tufano et al., 2024; Liu et al., 2025), undermining the reliability of past results.

Table 1 summarises the limitations of existing ACR evaluation benchmarks, underscoring the need for a new evaluation approach and dataset to reliably assess the capabilities of state-of-the-art LLMs. Our proposed **CodeReviewQA** benchmark focuses on addressing these gaps.

3 CodeReviewQA: MCQA Probes

Effective ACR relies heavily on the ability to comprehend R_{nl} under the context of H_{pre}. Rather than focusing this task as a sequence-to-sequence translation problem like the prior works, we argue that the model must be able to: 1) reason about the type of change R_{nl} is requesting and 2) identify the relevant lines of code in H_{pre} that is the subject of the change; and 3) formulate the required code changes from a wide action space of potential code edits that can be performed on H_{pre}, before generating the code revision H_{post}. The inability to perform the final code generation step may be caused by

Pre-Review Code Submission (H_{pre}):

```
1 from hypothesistooling . projects . hypothesispython import PYTHON_SRC
2 from hypothesistooling . scripts import pip_tool , tool_path
3
4 PYTHON_VERSIONS = [f"3.{v}" for v in range(7, 11)]
5
6 def test_mypy_passes_on_hypothesis () :
```

Code Review (R_{nl}): I think I'd prefer to write these out as literals, unless we can pull them out of the autoupdated CI config? Just thinking about how they'll stay up to date. I think we can also test against 3.11?

What type of change is the code review asking for?
A. Only add new lines of code
B. Only delete existing lines of code
C. Modify the code ✓

Which line numbers is the code review asking to modify code?
A. line number 1 B. line number 2
C. line number 4 ✓ D. line number 6

Which code revision is the code review asking for?

A.
```
4 − PYTHON_VERSIONS = [f"3.{v}" for v in range(7, 11)]
4 + PYTHON_VERSIONS >= ["3.7", "3.8", "3.9", "3.10", "3.11"]
```
B.
```
4 − PYTHON_VERSIONS = [f"3.{v}" for v in range(7, 11)]
4 + PYTHON_VERSIONS <= ["3.7", "3.8", "3.9", "3.10", "3.11"]
```
C.
```
4 − PYTHON_VERSIONS = [f"3.{v}" for v in range(7, 11)]
4 + PYTHON_VERSIONS != ["3.7", "3.8", "3.9", "3.10", "3.11"]
```
D. ✓
```
4 − PYTHON_VERSIONS = [f"3.{v}" for v in range(7, 11)]
4 + PYTHON_VERSIONS = ["3.7", "3.8", "3.9", "3.10", "3.11"]
```

Post-Review Code Revision (H_{post}):

```
1 from hypothesistooling . projects . hypothesispython import PYTHON_SRC
2 from hypothesistooling . scripts import pip_tool , tool_path
3
4 PYTHON_VERSIONS = ["3.7", "3.8", "3.9", "3.10", "3.11"]
5
6 def test_mypy_passes_on_hypothesis () :
```

Figure 2: The ACR task with intermediate reasoning steps presented as MCQA Probes.

any failure point amongst this multi-step reasoning process. Additionally, any failure within the intermediary reasoning steps might be propagated from a failure in a prior reasoning step, which obfuscates the specific incompetencies of the model.

To assess the proficiency of LLMs in ACR, we propose three MCQA probes, each representing a specific intermediate reasoning step. See Figure 2 for an example of the MCQA structure. Below, we describe the construction of each MCQA probe.

3.1 Change Type Recognition (CTR)

This is a closed set intent classification task that probes the model's ability to infer the intended type of code change. Specifically, given H_{pre}, the model must infer which general type of code change is being requested by R_{nl}. Formally, this problem requires the following estimation:

$$P(C_{type^+} | H_{pre}, R_{nl}) \tag{2}$$

where $C_{type^+} \in \{add, delete, modify\}$ denotes the correct code change type. There are three general types. Firstly, *add* requests involve only adding new lines of code. Secondly, *delete* requests involve only deleting existing lines of code. Lastly, *modify* requests involve altering the existing code by both deleting existing segments and adding new ones. The C_{type^-} distractors are the remaining two incorrect code change types.

This preliminary understanding serves as crucial conditional information that refines the problem space, providing the correct C_{type} context to subsequently locate where the code changes need to occur and identify what needs to be implemented.

3.2 Change Localisation (CL)

This is a coreference resolution task that probes the model's ability to locate where the intended code change is to occur. Specifically, given R_{nl}, the model must locate the precise lines of code within H_{pre} where the intended C_{type} code change should be applied. Formally, this problem requires the following estimation:

$$P(C_{loc^+} | H_{pre}, R_{nl}, C_{type}) \tag{3}$$

where C_{loc^+} denotes the exact set of line numbers that is the target of the intended code change. When $C_{type} \in \{delete, modify\}$, these are the exact lines of code that need to be deleted or modified. When $C_{type} = \{add\}$, these are the lines of code above where the new code needs to be added. The C_{loc^-} distractors are different sets of lines sampled from H_{pre}. We ensure $|C_{loc^-}| = |C_{loc^+}|$, such that set sizes do not reveal additional information.

As shown in Figure 2, natural code review comments often do not directly specify the exact location of the intended code change, rather this is implicitly conveyed based on a shared understanding between the reviewer and code author. Thus, the model must possess the ability to conduct anaphora resolution across modalities, between anaphors in R_{nl} and antecedents in H_{pre}. Inferring the incorrect C_{loc}, would subsequently hinder the model's ability to identify the H_{post} that accurately reflects the intended code change.

3.3 Solution Identification (SI)

This task probes the model's ability to both conduct open intent extraction from R_{nl} and identify the H_{post} that accurately reflects that intent. Given R_{nl}, the model must identify the correct H_{post} that reflects the intended C_{type} change on C_{loc} in H_{pre}. The intuition behind this task design is that if a model is able to generate a correct H_{post^+} revision, it should at least be able to identify that exact

H_{post+} solution amongst a solution space with incorrect H_{post-} alternatives. Formally, this problem requires the following estimation:

$$P(H_{post+}|H_{pre}, R_{nl}, C_{type}, C_{loc}) \quad (4)$$

where H_{post+} denotes the diff of the ground truth post-review code revision. We only include cases where $C_{type} \in \{add, modify\}$, as $\{delete\}$ cases merely delete C_{loc} located in the previous task.

3.4 Variation of Distractor Difficulty

The MCQA format allows flexibility in varying the difficulties of the distractors (i.e., the incorrect answer options). This not only allows us to stress test the models' level of understanding, but also enables the ability to evolve the benchmark against performance saturation. We specify the process of generating easy and hard distractors for *Change Localisation* and *Solution Identification*, as these tasks allow for variation in the solutions.

Change Localisation Distractors. We vary the difficulty based on the degree of overlap between the sets of the provided C_{loc} options. For the easy distractors, we sample C_{loc-} distractors from H_{pre}, such that the *Jaccard Similarity* between all answer options are as low as possible. This ensures that all answers are easy to distinguish from each other and the ground truth is more obvious to locate. For the hard distractors, we sample C_{loc-} distractors, such that the *Jaccard Similarity* between all answer options are as high as possible. This ensures that all answers are hard to discern from the ground truth, and requires the model to locate every exact line of the intended code change.

Solution Identification Distractors. To create distractor options H_{post-}, we generate modified versions of H_{post+} by perturbing code elements in C_{loc}, ensuring the intended code change is no longer correctly implemented. To create plausible but incorrect distractors that imitate possible mistakes that the models would make, we use a surrogate LLM [2] to 1) identify the code element with the highest average surprisal in H_{post+}, 2) mask it, and 3) retroactively fill the masks with diverse candidates. We keep candidates that are not equivalent to H_{post+} as valid H_{post-} distractor candidates. All generated distractors are manually verified for semantic in-equivalence to the ground truth. The algorithm for constructing H_{post-} distractors is illustrated in Algorithm 1 in the Appendix.

[2]We use a competitive surrogate LLM of code (Codestral-22B-v0.1) with a temperature of 3.5 to encourage diversity.

We vary the difficulty based on the degree of semantic similarity between the H_{post-} distractors and the H_{post+} ground truth. For the easy distractors, we retain the H_{post-} distractors which yield the lowest *cosine similarity* against H_{post+} in the embedding space of the surrogate model. This ensures that each H_{post-} is substantially different to H_{post+}, such that it is easy to discern. For the hard distractors, we retain the H_{post-} distractors which yield the highest *cosine similarity* against H_{post+}. This ensures that each H_{post-} is only marginally different from H_{post+}, such that it is hard to discern. See Figure 7 in the Appendix for examples of variation in difficulty.

4 Dataset Preparation

Data Source. We built our benchmark based on the most recently published automated code refinement dataset (Guo et al., 2024). This multilingual dataset was constructed from code reviews that occurred after January 1, 2022. To ensure that we have a sizable amount of clean data for each of the programming languages in our benchmark, we only include the nine most popular programming languages on GitHub i.e. C, C++, C#, Go, Java, Javascript, PHP, Python, Ruby. These 9,367 examples were mined from 259 repositories, filtered from a list of the most starred GitHub projects.

Data Sampling. To ensure diversity and quality in our benchmark, we conducted stratified sampling (Baltes and Ralph, 2022) across all nine programming languages in the dataset, and discarded any examples that were noisy or unfaithfully represented the task of code refinement. For each of the nine languages, we sampled until there were 100 clean examples each, resulting in 900 total examples in our benchmark. Within each language partition, we also conducted stratified sampling across projects to maintain diversity. This mitigated bias towards the code reviews of any specific project, the nature of which are influenced by their particular software development tools (Paschali et al., 2017), processes (Viggiato et al., 2019) and issues (Linares-Vásquez et al., 2014).

Data Curation. We discard examples that were noisy or unfaithfully represented the task of code refinement. The noisy examples refer to code review comments that are unclear, ignored, not asking for a code change, or linking to wrong code hunks. See Appendix A for a detailed explanation of these noise types. These kind of review comments were reported as critical quality issues with existing code review datasets by prior work (Tufano et al., 2024). Unfaithful examples refer to the scenarios that do not faithfully represent the ACR task i.e., reviews directly including the intended H_{post} revision implementation, reviews regarding simple code formatting, reviews that are not self-contained (Tufano et al., 2024; Lin et al., 2024). Instead, examples in the benchmark should represent meaningful quality improving code reviews that are beyond the capacity of traditional rule-based software engineering tools. See Appendix B for a detailed explanation of these unfaithful examples.

To discard noisy and unfaithful examples, we first applied heuristic filters as detailed in Appendix C, before manual verification. This resulted in 3,761 out of 9,367 examples being discarded from the source dataset. The manual discarding was conducted by the first and third authors, who are currently pursuing computer science PhDs focused on AI for software engineering. In the end, both annotators independently annotated 3k examples, and resolved all conflicts together across 46 rounds. The μ and σ of the Cohen's Kappa were 0.89 and 0.11, respectively. For C, JavaScript and Ruby, less than 100 clean examples could be obtained from the source dataset, thus, the remaining examples were sampled from code reviews conducted in 2021 (Li et al., 2022). The overall retention rate was 13%, highlighting critical quality issues in the source datasets, necessitating manual curation of ACR benchmarks for accurate and reliable evaluation. The final benchmark includes 199 of the original 259 GitHub repositories. Table 5 in the Appendix shows benchmark statistics.

5 Experimental Setup

5.1 MCQA Setup

To support the MCQA probes in **CodeReviewQA**, we detail our prompt design, answer extraction, and evaluation framework for invariance testing.

Prompt. We use multiple-choice prompting that takes an input containing three components: task definition, question, and options. The task definition specifies the broad purpose (e.g., "tests code review comprehension"). The question section presents the code review scenario within a template that includes programming language markers, H_{pre}, and R_{nl}. Finally, the options section lists multiple choice answers labelled alphabetically (A, B, C, D), with explicit instructions to respond with only the letter symbol. This prompt structure is used across all tasks, varying only the specific task parameters and answer options. See Figure 3 in the Appendix for all prompt templates used.

Answer Extraction. We use multiple choice prompting with a max output length of one, where the symbol token $\in \{A, B, C, D\}$ with the highest log probability is considered as the selected answer. This style of prompting avoids the conflation of likelihood of sequence and likelihood of answer, eliminates the need for normalisation and allows for direct comparison between answers (Robinson and Wingate, 2023). Our implementation uses the vLLM inference framework (Kwon et al., 2023) with guided decoding targeting the option symbols.

Invariant Test and Evaluation. To reduce the likelihood of random correct guesses, for each question, we exposed the models to every order combination of the answer options. This resulted in $N!$ runs per question, where N is the number of answer options provided. Thus, the likelihood of guessing the correct answer for all combinations of a question is merely $(\frac{1}{N})^{N!}$. With this, we assess the models' invariability in selecting the correct answer, regardless of the position of that answer. To be counted as correctly answering that question, the models must select the correct answer for all $N!$ runs, which is a more reliable indicator of the models' understanding (Wang et al., 2025).

5.2 Model Selection Criteria

We list the criteria that determines whether a LLM is appropriate for this benchmark.

MCQA Proficiency. The LLM must have achieved state-of-the-art results in MCQA style benchmarks e.g., MMLU (Hendrycks et al., 2021). This accounts for format as a confounding factor.

MCSB Proficiency. The LLM must demonstrate proficiency in multiple choice symbol binding (MCSB; Robinson and Wingate (2023)). This ensures that the answer extraction method is not a confounding factor. We report the Proportion of Plurality Agreement (PPA), which measures the degree of order invariance in selecting the symbol of the plurality answer. Formally, PPA is calculated as the average of $\frac{k}{N!}$ over a dataset, where k is the number of times the plurality answer's symbol yielded the highest log probability for a given question and $N!$ is the aforementioned number of order combinations for N answer options. MCSB proficiency is demonstrated when a PPA significantly higher than the random baseline of $\frac{1}{N}$ is achieved.

Coding Proficiency. In addition to understanding the natural language in R_{nl}, the model must also be able to understand the code in H_{pre} and H_{post}. Therefore, the LLM must have demonstrated proficiency in coding related benchmarks e.g., HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021).

In total, we selected 72 state-of-the-art open source LLMs, that have satisfied the three criteria. The included models are considered state-of-the-art as of March, 2025. See Table 7 in Appendix F for descriptions of all 72 models. The models are grouped into five scales based on their model parameters: ≤3B, ≤9B, ≤16B, ≤34B, and ≤72B. We selected models with ≤72B parameters, as it is the largest non-quantised model class that we can support locally to extract answer probabilities.

6 Results Overview

To compare how models of different scales perform on **CodeReviewQA**, we conducted experiments using all 72 models. Due to space limitations, complete results on ACR and all three MCQA probes are presented in Appendix F. In Table 2, we compare the MCQA probe results of the top-2 models in terms of ACR from each scale class.

Model	ACR	CTR	CL_E	CL_H	SI_E	SI_H
Qwen2.5-Coder-3B-Instruct	30.3	77.7	1.8	1.8	12.2	8.0
Llama-3.2-3B-Instruct	25.9	78.8	0.8	0.4	9.9	7.6
Qwen2.5-Coder-7B-Instruct	41.0	78.6	13.8	10.7	67.6	55.2
gemma-2-9b-it	39.0	74.1	59.2	52.0	58.8	49.6
Qwen2.5-Coder-14B-Instruct	46.6	73.9	46.7	37.3	65.5	56.2
phi-4	37.1	76.6	50.9	44.8	84.4	77.5
gemma-2-27b-it	46.4	74.0	70.1	58.7	76.2	65.7
Mistral-Small-Instruct-2409	45.6	76.7	38.4	31.6	63.8	60.1
Llama-3.1-70B-Instruct	**50.3**	68.4	**74.7**	**69.0**	84.2	76.7
Qwen2.5-72B-Instruct	48.7	**79.8**	64.2	58.3	**97.1**	**90.9**

ACR: Automated Code Refinement, **CTR:** Change Type Recognition
CL: Change Localisation, **SI:** Solution Identification, **E:** Easy, **H:** Hard

Table 2: MCQA Probe Accuracy (%) of top-2 models (per scale class) in terms of Exact Match Rate on ACR.

ACR vs MCQA Probes. Table 2 (column ACR) shows that Llama-3.1-70B-Instruct achieved the highest exact match rate of 50.3%. As expected, we find that larger LLMs tend to achieve higher exact match rates on average in ACR. However, the performance improvements appear to be diminishing when scaling past the ≤16B class of models. Specifically, we find that Llama-3.1-70B-Instruct only achieves a 3.7% increase over Qwen2.5-Coder-14B-Instruct, despite wielding a 5-fold increase in parameter count. Interestingly, we find that ACR performances are not always congruent with how models rank in MCQA probes. For example, Qwen2.5-72B-Instruct achieves an exact match rate approximately 2% lower than the top performer, but performs vastly better (> 10% increase) in terms of invariant accuracy in both CTR and SI. This type of inconsistency also occurs throughout the smaller scales. For example, Qwen2.5-Coder-14B-Instruct, gemma-2-27b-it, and Mistral-Small-Instruct-2409 achieve comparable exact match rates (≤ 1% difference) in ACR. However, their performances on CL and SI probes are substantially different. These additional insights highlight the benefits of intermediate feedback, beyond simple text matching evaluations on final code revision outputs. Below, we discuss results on the three MCQA probing tasks.

CTR Results. Table 2 (column CTR) shows that most of the ≤ 3B models were already competent in this task, with Llama-3.2-3B-Instruct achieving 78.8% invariant accuracy. Despite the promising results of small models, this ability plateaus as we scale model size. In fact, Qwen2.5-72B-Instruct only achieved a 1% improvement, despite having 24 times the amount of parameters. Interestingly, Llama-3.1-70B-Instruct even regresses in this capability, achieving the worst invariant accuracy across the top performing models. As knowledge of the general change type sets the entire context for deciding where and how to revise the code, the inability to improve on CTR presents a fundamental limit to downstream ACR performance.

CL Results. Table 2 (columns CL_E and CL_H) show that change localisation tends to be the most difficult comprehension task in the benchmark, where proficiency depends more heavily on scale. Table 10 in Appendix F shows the full results for CL_E, where most of the ≤ 3B models achieved invariant accuracies between 0%-3%. The only exceptions were Qwen2.5-3B-Instruct and Phi-3-mini-128k-instruct, which could achieve 39.3% and 34.1%, respectively. The discrepancy between ACR and CL results across the under parameterised models suggest a reliance on surface-level pattern recognition for ACR, that is not grounded in a robust understanding of the code review. In contrast, we find that many models from the ≤ 34B and ≤ 72B classes could achieve invariant accuracies of more than 70% for CL_E and more than 60% for CL_H. Nevertheless, proficiency in identifying exactly where the code revisions need to occur remains the largest challenge.

SI Results. Table 2 (columns SI_E and SI_H) show that proficiency in solution identification also strengthens with size. Despite this, Table 12 in Appendix F shows a few anomalies that outperform their scale class average by a large margin. For example, Phi-3-mini-128k-instruct can achieve an invariant accuracy of 58.2% for SI_E, whilst the majority of ≤3B models achieve less than 13%. In contrast, many models from the ≤72B class can achieve more than 80% for SI_E and more than 70% for SI_H. Most notably, Qwen2.5-72B-Instruct, could achieve a near perfect score of 97.1% for SI_E, and 90.9% for SI_H, despite previously achieving underwhelming results for change localisation. These results suggest that many larger models can identify the correct code revision at least under a drastically reduced solution space with semantically ambiguous distractors.

7 Insights from Probing ACR Failures

What does MCQA probes reveal about model failures in ACR? We use our MCQA probes to investigate ACR failure cases of the top performing models in Table 2. We define a failure as the case when the model does not generate a H_{post} revision that exactly matches the ground truth. For each failure case, we examine whether the model also fails on the MCQA probes. Through this analysis, we aim to identify specific weaknesses in the models' code review comprehension capabilities that may contribute to their ACR failures.

The results are presented in Table 3, including the percentage of failure on each of the MCQA probes, [3] and the overall failure on at least one probe (≥ 1 Probe). We find that all five models failed at least one MCQA probe for 76.5%-99.8% of the failure cases, indicating that the model struggled at the code review comprehension stage. CTR failures are seldom associated with the ACR failures, as this capability only accounts for 22.2%-37.4% of cases. In contrast, CL failures are often associated with ACR failures, particularly for the three smaller models, where 74.0%-99.4% of the cases overlap with ACR failures. SI failures account for 95.8% of cases for Qwen2.5-Coder-3B-Instruct. Thus, for the smallest model, most non-exact matches are associated with failures in both CL and SI. For both Qwen2.5-Coder-7B-Instruct and Qwen2.5-Coder-14B-Instruct, CL failure is the major factor associated with ACR failure. For

[3]For CL and SI, a failure in either difficulties is counted.

	%Fail			
Model	**≥ 1 Probe**	**CTR**	**CL**	**SI**
Qwen2.5-Coder-3B-Instruct	**99.8**	23.4	**99.4**	**95.8**
Qwen2.5-Coder-7B-Instruct	96.6	22.2	93.0	55.0
Qwen2.5-Coder-14B-Instruct	87.3	29.3	74.0	56.0
gemma-2-27b-it	75.5	25.7	53.7	41.4
Llama-3.1-70B-Instruct	76.5	**37.4**	46.3	32.7

CTR: Change Type Recognition, **CL:** Change Localisation, ≥ 1 **Probe:** Failed at least one probe, **SI:** Solution Identification

Table 3: MCQA Failure (%) in Non-Exact Match cases of the top performing model (per scale class) in ACR.

the larger models, gemma-2-27b-it and Llama-3.1-70B-Instruct, the probe failures are more evenly distributed, varying based on the model's specific incompetencies.

On the other hand, we also analysed the successful cases where the model achieves an exact match. Intuitively, if a model can achieve exact match on an example, it should be able to fully comprehend the code review, thus correctly answering all probes. However, the models could not accurately answer all probes for 49.0%-99.6% of their successful cases. This trend shows a strict inverse relationship with model size, where Llama-3.1-70B-Instruct and Qwen2.5-Coder-3B-Instruct could not consistently complete all of the probes for 49.0% and 99.6% of their successful cases, respectively. The ability to complete these ACR examples verbatim without a prerequisite understanding of the intent behind the code reviews allude to prior exposure and rote memorisation of these examples.

8 Evaluating Data Contamination

To what extent is CodeReviewQA resistant to data contamination? We utilise two canonical metrics for measuring data contamination, perplexity (Jelinek et al., 1977) and n-gram accuracy (Xu et al., 2024). Perplexity is an information-theoretic metric, which quantifies the uncertainty of a language model in a token sequence (Jelinek et al., 1977), which can be formulated as:

$$\text{PPL}(\mathbf{X}) = \exp\left(-\frac{1}{t}\sum_{t=0}^{t}\log p_\theta(x_i|x_{<i})\right) \quad (5)$$

where $\mathbf{X} = [x_0, x_1, ..., x_t]$ denotes a tokenised sequence. In our case, the sequence is a concatenation of both the prompt (including H_{pre}, R_{nl}) and the solution. For ACR, the solution is H_{post}, and for the MCQA probes, it is the answer options. A low perplexity score indicates high confidence, whilst a high perplexity score indicates

Benchmark	Format	Llama-3.1-70B		Qwen2.5-72B	
		PPL	NG_5	PPL	NG_5
CodeReviewer	ACR	4.1	28.1	3.6	30.7
CodeReview-New	ACR	4.4	40.3	3.9	42.6
CodeReviewQA	ACR	4.5	40.3	4.1	42.0
	MCQA	**6.0**	**25.1**	**5.4**	**26.8**

ACR: Automated Code Refinement, **NG_5:** 5-gram Accuracy
MCQA: Multiple Choice Question & Answer, **PPL:** Perplexity Scores

Table 4: Perplexity Scores and 5-gram Accuracy (%) of **CodeReviewQA** against existing ACR benchmarks.

low confidence. Unusually low perplexity scores may indicate data contamination. N-gram accuracy measures the model's ability to predict random n-gram sequences from K starting points that are uniformly sampled from an example (Xu et al., 2024), i.e., the aforementioned sequence X. It is calculated by the following equation:

$$\mathrm{NG}(\mathbf{X}) = \frac{1}{\eta \cdot K} \sum_{i=0}^{\eta} \sum_{j=0}^{K} I(X_{s_j:s_j+n}, \hat{X}_{s_j:s_j+n}) \tag{6}$$

where η denotes the corpus size, i denotes the i_{th} sequence in the corpus, s_j denotes the index of the j^{th} starting point, $X_{s_j:s_j+n}$ denotes the ground truth n-gram to be predicted and I denotes an indicator function that applies exact match. Unusually high n-gram accuracies may indicate data contamination. Following prior work (Xu et al., 2024), we set K = 5 and $n = 5$ to measure 5-gram accuracy.

For this experiment, we use the largest and newest models that we can support from the most popular model families, as they are most likely to exhibit memorisation (Kiyomaru et al., 2024). We use base versions of models as instruction-tuning optimises for responses to prompts rather than completing sequences verbatim. To test the effectiveness of MCQA reformulation in mitigating data contamination, we compare our benchmark in MCQA probe form with the original ACR form, as well as the most widely used ACR benchmarks, i.e. CodeReviewer (Li et al., 2022) and CodeReview-New (Guo et al., 2024).

Table 4 shows the perplexity and 5-gram accuracies on the three benchmarks, based on two popular base models Llama-3.1-70B and Qwen2.5-72B. We find that perplexity on the older CodeReviewer benchmark is far lower than on CodeReview-New, yet the 5-gram accuracies are also lower. A likely explanation is that older code reviews may have been extensively included in the models' training phase and therefore reflect the predominant patterns in their learned distribution, however, since they may not have been included in the latter stages, there is less verbatim memorisation of the examples (Kiyomaru et al., 2024). In contrast, the newer code reviews may represent a distribution shift, yet is more likely to be included in the latter stages of training, thus concurrently increasing both perplexity and verbatim memorisation at the same time.

We find that **CodeReviewQA** in the original ACR format yields similar results to CodeReview-New, which is within expectation as one is simply a curated subset of the other. However, when reformulating into the MCQA probe format, our benchmark yields significantly higher perplexity than all past benchmarks with lower 5-gram accuracies, despite using the same examples. Therefore, we find that MCQA reformulation with synthetic questions and answers does mitigate the effects of data contamination, allowing for the reuse of code reviews that may have been previously included in the training corpus. Coinciding with our experimental results, models that perform well on ACR with only memorisation can be exposed when evaluated with MCQA probes on the same examples.

9 Conclusion

In this study, we focus on evaluating recent large language models' capabilities in automated code refinement, a challenging task that requires an understanding of the intended code revisions behind natural language code reviews, before subsequently performing them. We addressed two key limitations in existing work, the inability of text matching metrics to provide fine-grained feedback on intermediate model failures and the potential for benchmark contamination. To this end, we propose **CodeReviewQA**, which consists of 900 manually curated high-quality code review examples. We reformulated the generative task of automated code refinement into three intermediate reasoning probes, which are presented in the multiple choice question and answering format. Our experimental results across 72 state-of-the-art large language models revealed capability differences that traditional evaluation metrics failed to capture. Additionally, our evaluation of data contamination demonstrated that task reformulation effectively mitigates these concerns, exposing cases of memorisation without comprehension.

10 Limitations

Whilst **CodeReviewQA** advances the evaluation of automated code refinement, it still faces limitations.

Size of dataset. Our benchmark has a relatively modest size due to the difficulty of scaling rigorous manual verification. Despite this, the benchmark was designed to be diverse and comprehensive through stratified sampling, covering real-world code reviews from 199 different GitHub projects in nine of the most popular programming languages.

Construction of distractors. The change localisation task focused on line level localisation rather than a more fine-grained level (e.g., token level). However, our findings show that many models struggle with identifying the location of changes even at the line level. Future work can further explore approaches to automatically construct and evaluate localisation at the token level. The solution identification task relies on a competitive surrogate LLM for constructing challenging distractors. Whilst Codestral-22B-v0.1 is considered competitive at the time of this writing, benchmark saturation remains inevitable as more powerful LLMs are developed. Nonetheless, the benchmark can be evolved by swapping to a more up-to-date surrogate model in the future, thus increasing the difficulty and relevance of the distractors.

Interaction among capabilities. As we were interested in analysing the successive code review comprehension capabilities in isolation, each probe was designed to be independent of each other by including the preceding probe's ground truth as conditional information. We did not investigate the causal relationships between the capabilities tested in the probes, meaning that failure in one probe does not predict performance on another. However, our experimental results demonstrate that analysing disentangled capabilities alongside the automated code refinement task provides more interpretable insights into model weaknesses.

Diversity of prompts. We used the same prompt and hyperparameters for each task to maintain consistency and comparability across models. Prompt variation might impact model performance differently. However, our main focus was not to find the optimal prompt for each model, but to gauge their systematic differences across different capabilities required for automated code refinement.

Semantic level data contamination. Our MCQA reformulation process only mitigates the effects of surface level data contamination i.e., when examples are presented as a next token prediction task in their original form. Given that the actual content of the examples have not been altered, there still exists the risk of data contamination at the semantic level, where models can leverage the learned meanings conveyed in the code reviews that they have been exposed to. To the best of our knowledge, there are no viable solutions that can completely address this concern, apart from collecting new code reviews or using completely private projects, which poses concerns of feasibility and generalisability. Nevertheless, our benchmark evaluates knowledge invariance in comprehending and addressing code reviews. Assessing this type of generalisation of the contaminated examples can still be considered a meaningful measurement of learned capabilities, beyond simple memorisation.

References

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program synthesis with large language models. *CoRR*, abs/2108.07732.

Sebastian Baltes and Paul Ralph. 2022. Sampling in software engineering research: a critical review and guidelines. *Empirical Software Engineering*, 27(4):94.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Vasiliki Efstathiou and Diomidis Spinellis. 2018. Code review comments: language matters. In *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results, ICSE (NIER) 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pages 69–72. ACM.

Qi Guo, Junming Cao, Xiaofei Xie, Shangqing Liu, Xiaohong Li, Bihuan Chen, and Xin Peng. 2024. Exploring the potential of chatgpt in automated code refinement: An empirical study. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*, pages 34:1–34:13. ACM.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2018. Deep code comment generation. In *Proceedings of the 26th Conference on Program Comprehension, ICPC 2018, Gothenburg, Sweden, May 27-28, 2018*, pages 200–210. ACM.

Frederick Jelinek, Robert L. Mercer, Lalit R. Bahl, and Janet M. Baker. 1977. Perplexity—a measure of the difficulty of speech recognition tasks. *Journal of the Acoustical Society of America*, 62.

Siyuan Jiang, Ameer Armaly, and Collin McMillan. 2017. Automatically generating commit messages from diffs using neural machine translation. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*, pages 135–146. IEEE.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Hirokazu Kiyomaru, Issa Sugiura, Daisuke Kawahara, and Sadao Kurohashi. 2024. A comprehensive analysis of memorization in large language models. In *Proceedings of the 17th International Natural Language Generation Conference, INLG 2024, Tokyo, Japan, September 23 - 27, 2024*, pages 584–596. ACL.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pages 611–626. ACM.

VI Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Proceedings of the Soviet physics doklady*.

Zhiyu Li, Shuai Lu, Daya Guo, Nan Duan, Shailesh Jannu, Grant Jenks, Deep Majumder, Jared Green, Alexey Svyatkovskiy, Shengyu Fu, and Neel Sundaresan. 2022. Automating code review activities by large-scale pre-training. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*, pages 1035–1047. ACM.

Hong Yi Lin, Patanamon Thongtanunam, Christoph Treude, and Wachiraphan Charoenwet. 2024. Improving automated code reviews: Learning from experience. In *21st IEEE/ACM International Conference on Mining Software Repositories, MSR 2024, Lisbon, Portugal, April 15-16, 2024*, pages 278–283. ACM.

Mario Linares-Vásquez, Sam Klock, Collin McMillan, Aminata Sabané, Denys Poshyvanyk, and Yann-Gaël Guéhéneuc. 2014. Domain matters: bringing further evidence of the relationships among anti-patterns, application domains, and quality-related metrics in java mobile apps. In *22nd International Conference on Program Comprehension, ICPC 2014, Hyderabad, India, June 2-3, 2014*, pages 232–243. ACM.

Chunhua Liu, Hong Yi Lin, and Patanamon Thongtanunam. 2025. Too noisy to learn: Enhancing data quality for code review comment generation. In *22nd IEEE/ACM International Conference on Mining Software Repositories, MSR 2025, Ottawa, Canada, April 28-29, 2025*. IEEE.

Zhongxin Liu, Xin Xia, Christoph Treude, David Lo, and Shanping Li. 2019. Automatic generation of pull request descriptions. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*, pages 176–188. IEEE.

Junyi Lu, Lei Yu, Xiaojia Li, Li Yang, and Chun Zuo. 2023. Llama-reviewer: Advancing code review automation with large language models through parameter-efficient fine-tuning. In *34th IEEE International Symposium on Software Reliability Engineering, ISSRE 2023, Florence, Italy, October 9-12, 2023*, pages 647–658. IEEE.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, ACL 2002, Philadelphia, PA, USA, July 6-12, 2002*, pages 311–318. ACL.

Maria Eleni Paschali, Apostolos Ampatzoglou, Stamatia Bibi, Alexander Chatzigeorgiou, and Ioannis Stamelos. 2017. Reusability of open source software across domains: A case study. *Journal of Systems and Software*, 134:211–227.

Chanathip Pornprasit and Chakkrit Tantithamthavorn. 2024. Fine-tuning and prompt engineering for large language models-based code review automation. *Information and Software Technology*, 175:107523.

Mohammad Masudur Rahman, Chanchal K. Roy, and Raula Gaikovina Kula. 2017. Predicting usefulness of code review comments using textual features and developer experience. In *Proceedings of the 14th International Conference on Mining Software Repositories, MSR 2017, Buenos Aires, Argentina, May 20-28, 2017*, pages 215–226. IEEE.

Joshua Robinson and David Wingate. 2023. Leveraging large language models for multiple choice question answering. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Ripon K. Saha, Yingjun Lyu, Wing Lam, Hiroaki Yoshida, and Mukul R. Prasad. 2018. Bugs.jar: a large-scale, diverse dataset of real-world java bugs. In *Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018*, pages 10–13. ACM.

June Sallou, Thomas Durieux, and Annibale Panichella. 2024. Breaking the silence: the threats of using llms in software engineering. In *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results, NIER@ICSE 2024, Lisbon, Portugal, April 14-20, 2024*, pages 102–106. ACM.

Patanamon Thongtanunam, Chanathip Pornprasit, and Chakkrit Tantithamthavorn. 2022. Autotransform: Automated code transformation to support modern code review process. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*, pages 237–248. ACM.

Michele Tufano, Jevgenija Pantiuchina, Cody Watson, Gabriele Bavota, and Denys Poshyvanyk. 2019. On learning meaningful code changes via neural machine translation. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, pages 25–36. IEEE / ACM.

Rosalia Tufano, Ozren Dabic, Antonio Mastropaolo, Matteo Ciniselli, and Gabriele Bavota. 2024. Code review automation: Strengths and weaknesses of the state of the art. *IEEE Transactions on Software Engineering*, 50(2):338–353.

Rosalia Tufano, Simone Masiero, Antonio Mastropaolo, Luca Pascarella, Denys Poshyvanyk, and Gabriele Bavota. 2022. Using pre-trained models to boost code review automation. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*, pages 2291–2302. ACM.

Rosalia Tufano, Luca Pascarella, Michele Tufano, Denys Poshyvanyk, and Gabriele Bavota. 2021. Towards automating code review activities. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*, pages 163–174. IEEE.

Markos Viggiato, Johnatan Oliveira, Eduardo Figueiredo, Pooyan Jamshidi, and Christian Kästner. 2019. Understanding similarities and differences in software development practices across domains. In *Proceedings of the 14th International Conference on Global Software Engineering, ICGSE 2019, Montreal, QC, Canada, May 25-31, 2019*, pages 74–84. IEEE / ACM.

Haochun Wang, Sendong Zhao, Zewen Qiang, Nuwa Xi, Bing Qin, and Ting Liu. 2025. Llms may perform MCQA by selecting the least incorrect option. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, pages 5852–5862. ACL.

Ruijie Xu, Zengzhi Wang, Run-Ze Fan, and Pengfei Liu. 2024. Benchmarking benchmark leakage in large language models. *CoRR*, abs/2404.18824.

Lanxin Yang, Jinwei Xu, Yifan Zhang, He Zhang, and Alberto Bacchelli. 2023. Evacrc: Evaluating code review comments. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, San Francisco, CA, USA, December 3-9, 2023*, pages 275–287. ACM.

Wenhong Zhu, Hongkun Hao, Zhiwei He, Yunze Song, Jiao Yueyang, Yumeng Zhang, Hanxu Hu, Yiran Wei, Rui Wang, and Hongyuan Lu. 2024. CLEAN-EVAL: clean evaluation on contaminated large language models. In *Findings of the Association for Computational Linguistics: NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 835–847. ACL.

Terry Yue Zhuo, Vu Minh Chien, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen GONG, James Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, David Lo, Binyuan Hui, Niklas Muennighoff, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. 2025. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, Singapore, Apr 24-28, 2025*. OpenReview.net.

A Data Quality Issue Details

We explain the four types of data quality issues found in code review datasets. Firstly, *unclear comments* are review comments where even humans cannot comprehend the intended change. Secondly, *no change asked* refers to review comments that are not actionable. Thirdly, *ignored comment* are examples where the developer ignores the review comment, resulting in a post-review code revision H_{post} that does not reflect the intended code change. Lastly, *wrong linking* refers to data mining issues, where the review comment is not related to the paired pre-review code submission H_{pre}.

B Unfaithful Example Details

We explain the three types of unfaithful examples found in code review datasets. Firstly, some code reviews directly include the *intended code revision implementation*. These cases can be resolved by directly copy and pasting from the review itself, which does not assess natural language comprehension. Secondly, *code formatting* related examples can already be resolved by linters and therefore are not useful to learn. These examples also fail to assess the models' ability in handling challenging and meaningful code reviews. Thirdly, code reviews that are *not self-contained* require information beyond the provided code hunk H_{pre} to understand, thus, it is impossible for the model (or even a human) to intuit the intended code change.

C Heuristic Filtering Details

We conducted keyword-based filtering to automatically discard examples that clearly violate the data quality and faithfulness issues mentioned above. With regards to *unclear comments*, we discarded reviews with less than 10 characters, since they are likely to be too short to convey a code change requirement. With regards to code reviews that already contain the *intended code revision implementation*, we discarded reviews that included the "“‘" GitHub code block indicator. With regards to code reviews that are only demanding *code formatting* changes, we removed reviews that mention "indentation", "spacing" and "lint". With regards to reviews that are *not self-contained*, we removed reviews that mention "revert", "as above" and "ditto". The purpose of this step was to reduce human workload in the proceeding manual discarding process.

D Descriptive Statistics

We explain the descriptive statistics used in this study for describing the benchmark.

Comment length is measured by the number of whitespace separated words in the code review comment. Longer comments may contain more complex requirements, explanations or other discussions. Table 5 shows that the average comment length is 18 words. See Figure 8 for examples of different comment lengths.

Code edit distance represents the size of the code change between H_{pre} and H_{post}. Given that some examples only involve changes in the inlined code comment, we use the more general Levenshtein distance (Levenshtein, 1966) to measure the number of character edits between the two versions of code. Table 5 shows that the average code edit distance is 56 characters. See Figure 9 for examples of different code edit distances.

Statistic	Min	Max	Mean	SD	Q1	Median	Q3
Comment Length	2	98	18	15	8	13	23
Code Edit Distance	2	827	56	85	9	25	67
Change Locations							
Add (35)	1	2	1	0	1	1	1
Delete (144)	1	19	3	4	1	2	4
Modify (721)	1	15	2	2	1	1	2
Code Element Ratio	0.00	0.89	0.09	0.15	0.00	0.00	0.13
Specification Ratio	0.04	165.40	4.88	9.79	0.67	1.83	5.07

See Appendix D for a detailed explanation of the descriptive statistics.

Table 5: **CodeReviewQA** descriptive statistics.

Change locations is the number of lines involved in the change, as discussed in the task of change localisation. Table 5 shows the change location statistics. For the 35 (4% of total) code reviews that request to *add* code, the average number of change locations is 1 line. For the 144 (16% of total) code reviews that request to *delete* code, the average number of change locations is 3 lines. For the 721 (80% of total) code reviews that request to *modify* code, the average number of change locations is 2 lines.

Code element ratio is the proportion of tokens in the code review comment that are code elements. It is calculated as $\frac{\text{Code elements}}{\text{Comment length}}$. Reviewers may use code elements in conjunction with natural language to describe the intended code change. Comments with a higher proportion of code tokens may be more explicit in their specification of the requirements (Rahman et al., 2017). Table 5 shows that the average code element ratio is 0.09. See Figure 10 for examples of different code element ratios.

Specification ratio is the code edit distance of the change divided by the length of its respective code review comment. It is calculated as $\frac{\text{Code edit distance}}{\text{Comment length}}$, the number of character edits with respect to each word in the comment. Since code review comments may be under-specified and implicit, we use specification ratio as a heuristic metric that incorporates this notion. Intuitively, examples with larger code edit distances and shorter comment lengths i.e. larger specification ratios, may be under-specified in its description of the required code change. Table 5 shows that the average specification ratio is 4.88. See Figure 11 for examples of different specification ratios.

E Implementation Details

The experiments were carried out on a single node consisting of 64 cores (Intel(R) Xeon(R) Platinum 8462Y+ @ 2.80GHz), 928GBs of RAM and four GPUs (NVIDIA H100-80GB SXM5). For efficient inference, we utilised the default vLLM implementation with Hugging Face models. [4] For all tasks, we set the temperature to zero for greedy decoding.

F Experimental Results on 72 models

This section shows our complete results. The details of all evaluated LLMs are presented in Table 7, and their achieved exact match rate on ACR in Table 8. For the MCQA probes, we report both their proportion of plurarity agreement and invariant accuracy. If the achieved proportion is far higher than the stated random probability on any probe, the model is considered to have multiple choice symbol binding proficiency. Table 9 shows the full results on CTR. Table 10 and Table 11 shows full results on the easy and hard variations of CL, respectively. Table 12 and Table 13 shows full results on the easy and hard variations of SI, respectively.

G Advanced Prompting

Since our main results were based on zero-shot prompting, we further investigated the effects of advanced prompting i.e., few-shot, chain-of-thought, on MCQA probe performance. Specifically, we were interested if advanced prompting could increase the performance of the top performing model i.e., Llama-3.1-70B-Instruct. For few-shot prompting, we considered both one-shot and two-shot scenarios with real examples held out from the benchmark. For the few-shot prompt templates, see Figure 4 for CTR, Figure 5 for CL and Figure 6 for SI. For chain-of-thought, we only considered zero-shot due to the lack of real reasoning traces to use as examples. This was invoked by appending "*Let's think step by step*" to the end of the zero-shot prompts and instructing the model to finalise their answer with "*The final answer is*".

The results are shown in Table 6. Overall, we find that advanced prompting does not outperform the zero-shot strategy. For CTR, we find that zero-shot performed similar to chain-of-thought, where both accuracies were approximately 68%. In contrast, the few-shot methods could achieve 74.1-76.9%, where two-shot prompting was the top performing prompt strategy. In terms of CL easy, we find that one-shot and chain-of-thought were the worst performers, achieving 68.6% and 65.8%, respectively. In contrast, zero-shot and two-shot performed similarly, achieving 74.7% and 75.8%, respectively. For the remaining MCQA probes, all advanced prompting techniques underperformed against the zero-shot strategy, especially for SI.

Whilst chain-of-thought consistently degraded the model's performance on the MCQA probes, few-shot prompting showed promise. Specifically, two-shot prompting was the top performing strategy in two of the five probes, without optimising for the best examples. We did not optimise this selection due to the lack of clean code review examples. As a result, we only used two held out examples collected during the manual curation process. Future researchers can investigate the effects of advanced example selection e.g., retrieval-augmented generation, for few-shot prompting strategies.

Prompt Strategy	CTR	CL_E	CL_H	SI_E	SI_H
Zero-Shot	68.4	74.7	**69.0**	**84.2**	**76.7**
One-Shot	74.1	68.6	61.9	59.9	55.6
Two-Shot	**76.9**	**75.8**	68.7	57.3	54.6
Chain-of-Thought	68.6	65.8	60.1	66.6	60.0

CTR: Change Type Recognition, **E:** Easy, **H:** Hard
CL: Change Localisation, **SI:** Solution Identification

Table 6: MCQA Probe Accuracy (%) of Llama-3.1-70B-Instruct w/ Advanced Prompting.

[4] https://huggingface.co/docs/hub/en/models-the-hub

Algorithm 1: Create H_{post^-} Distractors for Solution Identification

Input: Surrogate LLM f_θ, Temperature k, No. of Distractors N, Difficulty ϕ

Output: Set of H_{post^-} Distractors D

`// Identify code elements in the changed lines of the post-review code revision`

$Lines \leftarrow \text{GetChangedLines}(H_{post^+})$;

$AST \leftarrow \text{GetAbstractSyntaxTree}(H_{post^+})$;

$Nodes \leftarrow \text{GetLeafNodes}(AST, Lines)$;

`// Calculate average token surprisal for each identified code element`

$S_{token}, S_{node}, C_{distractor}, D \leftarrow \emptyset, \emptyset, \emptyset, \emptyset$;

for $c_t \in H_{post^+}$ **do**

  $\boldsymbol{h}_{t-1} \leftarrow f_\theta(H_{pre}, R_{nl}, c_{<t})$;

  $S_{token}[c_t] \leftarrow -\log_2 P(c_t | \boldsymbol{h}_{t-1})$;

for $n_t \in Nodes$ **do**

  for $c_t \in n_t$ **do**

    $S_{node}[n_t] \leftarrow S_{node}[n_t] + S_{token}[c_t]$;

  $S_{node}[n_t] \leftarrow \frac{S_{node}[n_t]}{length(n_t)}$;

`// Mask the code element with the highest average token surprisal`

$n_{max} \leftarrow \text{GetMaxKeys}(S_{node}, 1)$;

$Mask \leftarrow \text{ApplyMask}(H_{post^+}, n_{max})$

`// Create a set of distractors for each of the easy and hard variations`

while $length(C_{distractor}) < 2 \times N$ **do**

  $\hat{n}_{max} \leftarrow \arg\max_{n_{max}} P_k(n_{max} | f_\theta(Mask))$;

  $Candidate = \text{InFill}(Mask, \hat{n}_{max})$;

  if $Candidate \neq H_{post^+} \wedge Candidate \notin C_{distractor}$ **then**

    $\theta = \text{Cosine}(f_\theta(Candidate), f_\theta(H_{post^+}))$;

    $C_{distractor}[Candidate] \leftarrow \theta$;

`// Select distractors most semantically different from ground truth for easy and distractors most semantically similar to ground truth for hard`

if $\phi = easy$ **then**

  $D \leftarrow \text{GetMinKeys}(C_{distractor}, N)$;

else if $\phi = hard$ **then**

  $D \leftarrow \text{GetMaxKeys}(C_{distractor}, N)$;

Automated Code Refinement (ACR)
{lang} = C/CPP/CSharp/Go/Java/JavaScript/PHP/Python/Ruby
The following {lang} code snippet has received a code review.
[{lang}]
{code_snippet}
[/{lang}]
[CODE REVIEW]
{code_review}
[/CODE REVIEW]
Please generate a revised version of the code snippet according to the code review. Do not add explanations.
[{lang}]

Change Type Recognition (CTR)
{option_a}, {option_b}, {option_c} = only add new lines of code/only delete existing lines of code/modify the code
The following is a multiple choice question (with answers) that tests code review comprehension.
Question: Given this {lang} code snippet, what type of change is the code review asking for?
[{lang}]
{code_snippet}
[/{lang}]
[CODE REVIEW]
{code_review}
[/CODE REVIEW]
Possible answers:
A. {option_a}
B. {option_b}
C. {option_c}
Answer with the letter symbol only. Answer:

Change Localisation (CL)
{change_type} = add new lines of code under/delete code/modify code
The following is a multiple choice question (with answers) that tests code review comprehension.
Question: Given this {lang} code snippet, which line numbers is the code review asking to {change_type}?
[{lang}]
{code_snippet}
[/{lang}]
[CODE REVIEW]
{code_review}
[/CODE REVIEW]
Possible answers:
A. {option_a}
B. {option_b}
C. {option_c}
D. {option_d}
Answer with the letter symbol only. Answer:

Solution Identification (SI)
The following is a multiple choice question (with answers) that tests code review comprehension.
Question: Given this {lang} code snippet, which code revision is the code review asking for?
[{lang}]
{code_snippet}
[/{lang}]
[CODE REVIEW]
{code_review}
[/CODE REVIEW]
Possible answers:
A. {option_a}
B. {option_b}
C. {option_c}
D. {option_d}
Answer with the letter symbol only. Answer:

Figure 3: Zero-Shot Prompt Templates.

Change Type Recognition (CTR)

The following are multiple choice questions (with answers) that tests code review comprehension.
Question: Given this Java code snippet, what type of change is the code review asking for?

[Java]

```
private void launchZoomActivityAfterPermissionCheck(final View view) {
        final Context ctx = view.getContext();
        final Intent zoomableIntent = new Intent(ctx, ZoomableActivity.class);
        zoomableIntent.setData(Uri.parse(media.getImageUrl()));
        zoomableIntent.putExtra("Origin", "MediaDetail");
        ctx.startActivity(
            zoomableIntent
        );
```

[/Java]

[CODE REVIEW]

Could you please rename to "MediaDetails"?

[/CODE REVIEW]

Possible answers:

A. only add new lines of code

B. only delete existing lines of code

C. modify the code

Answer with the letter symbol only. Answer:

C

Question: Given this CSharp code snippet, what type of change is the code review asking for?

[CSharp]

```
public static class DateTimeDefinitions
        public const string QuarterTypeRegex = @"(trimestral(mente)?)$";
        public const string SemiAnnualTypeRegex = @"(semestral(mente)?)$";
        public const string YearTypeRegex = @"(anual(mente)?)$";
        public static readonly IList<string> FutureTerms = new List<string>
        {
            @"esea"
        };
    }
}
\ No newline at end of file
```

[/CSharp]

[CODE REVIEW]

There are two issues here. One, this term is not a "future" term, so we need a better name.
Second, this value is incorrect in PT, it should be "esse", "essa", "este", "esta".

[/CODE REVIEW]

Possible answers:

A. only add new lines of code

B. only delete existing lines of code

C. modify the code

Answer with the letter symbol only. Answer:

C

Question: Given this {lang} code snippet, what type of change is the code review asking for?

[{lang}]

{code_snippet}

[/{lang}]

[CODE REVIEW]

{code_review}

[/CODE REVIEW]

Possible answers:

A. only add new lines of code

B. only delete existing lines of code

C. modify the code

Answer with the letter symbol only. Answer:

Figure 4: Few-Shot CTR Prompt Template.

Change Localisation (CL)

{change_type} = add new lines of code under/delete code/modify code

The following are multiple choice questions (with answers) that tests code review comprehension.
Question: Given this Java code snippet, which line numbers is the code review asking to modify code?

[Java]

```
1   private void launchZoomActivityAfterPermissionCheck(final View view) {
2           final Context ctx = view.getContext();
3           final Intent zoomableIntent = new Intent(ctx, ZoomableActivity.class);
4           zoomableIntent.setData(Uri.parse(media.getImageUrl()));
5           zoomableIntent.putExtra("Origin", "MediaDetail");
6           ctx. startActivity (
7               zoomableIntent
8           );
```

[/Java]

[CODE REVIEW]

Could you please rename to "MediaDetails"?

[/CODE REVIEW]

Possible answers:

A. 3

B. 4

C. 5

D. 6

Answer with the letter symbol only. Answer:

C

Question: Given this CSharp code snippet, which line numbers is the code review asking to modify code?

[CSharp]

```
1     public static class DateTimeDefinitions
2             public const string QuarterTypeRegex = @"(trimestral(mente)?)$";
3             public const string SemiAnnualTypeRegex = @"(semestral(mente)?)$";
4             public const string YearTypeRegex = @"(anual(mente)?)$";
5             public static readonly IList<string> FutureTerms = new List<string>
6             {
7                 @"esea"
8             };
9         }
10    }
11\ No newline at end of file
```

[/CSharp]

[CODE REVIEW]

There are two issues here. One, this term is not a "future" term, so we need a better name.
Second, this value is incorrect in PT, it should be "esse", "essa", "este", "esta".

[/CODE REVIEW]

Possible answers:

A. 5, 7

B. 2, 3

C. 1, 4

D. 8, 11

Answer with the letter symbol only. Answer:

A

Question: Given this {lang} code snippet, which line numbers is the code review asking to {change_type}?

[{lang}]

{code_snippet}

[/{lang}]

[CODE REVIEW]

{code_review}

[/CODE REVIEW]

Possible answers:

A. {option_a}

B. {option_b}

C. {option_c}

D. {option_d}

Answer with the letter symbol only. Answer:

Figure 5: Few-Shot CL Prompt Template.

Solution Identification (SI)

The following are multiple choice questions (with answers) that tests code review comprehension.
Question: Given this Java code snippet, which code revision is the code review asking for?

[Java]
```
private void launchZoomActivityAfterPermissionCheck(final View view) {
    final Context ctx = view.getContext();
    final Intent zoomableIntent = new Intent(ctx, ZoomableActivity.class);
    zoomableIntent.setData(Uri.parse(media.getImageUrl()));
    zoomableIntent.putExtra("Origin", "MediaDetail");
    ctx. startActivity (
        zoomableIntent
    );
```
[/Java]

[CODE REVIEW]

Could you please rename to "MediaDetails"?

[/CODE REVIEW]

Possible answers:

A.
```
5− zoomableIntent.putExtra("Origin", "MediaDetail"); {    5+ zoomableIntent.postExtra("Origin", "MediaDetails");
```
B.
```
5− zoomableIntent.putExtra("Origin", "MediaDetail"); {    5+ zoomableIntent.getExtra("Origin", "MediaDetails");
```
C.
```
5− zoomableIntent.putExtra("Origin", "MediaDetail"); {    5+ zoomableIntent.putExtra("Origin", "MediaDetails");
```
D.
```
5− zoomableIntent.putExtra("Origin", "MediaDetail"); {    5+ zoomableIntent.renameExtra("Origin", "MediaDetails");
```

Answer with the letter symbol only. Answer:

C

Question: Given this CSharp code snippet, which code revision is the code review asking for?

[CSharp]
```
public static class DateTimeDefinitions
    public const string QuarterTypeRegex = @"(trimestral(mente)?)$";
    public const string SemiAnnualTypeRegex = @"(semestral(mente)?)$";
    public const string YearTypeRegex = @"(anual(mente)?)$";
    public static readonly IList<string> FutureTerms = new List<string>
    {
        @"esea"
    };
  }
}
\ No newline at end of file
```
[/CSharp]

[CODE REVIEW]

There are two issues here. One, this term is not a "future" term, so we need a better name.
Second, this value is incorrect in PT, it should be "esse", "essa", "este", "esta".

[/CODE REVIEW]

Possible answers:

A.
```
5− public static readonly IList<string> FutureTerms = new List<string>
5+ public static readonly IList<string> ThisTerms = new List<string>
7− @"esea" 7+ @"esse" 8+ @"essa" 9+ @"este" 10+ @"esta"
```
B.
```
5− public static readonly IList<string> FutureTerms = new List<string>
5+ public static readonly IList<string> NotTerms = new List<string>
7− @"esea" 7+ @"esse" 8+ @"essa" 9+ @"este" 10+ @"esta"
```
C.
```
5− public static readonly IList<string> FutureTerms = new List<string>
5+ public static readonly IList<string> futureTerms = new List<string>
7− @"esea" 7+ @"esse" 8+ @"essa" 9+ @"este" 10+ @"esta"
```
D.
```
5− public static readonly IList<string> FutureTerms = new List<string>
5+ public static readonly IList<string> betterTerms = new List<string>
7− @"esea" 7+ @"esse" 8+ @"essa" 9+ @"este" 10+ @"esta"
```

Answer with the letter symbol only. Answer:

A

Question: Given this {lang} code snippet, which code revision is the code review asking for?

[{lang}]

{code_snippet}

[/{lang}]

[CODE REVIEW]

{code_review}

[/CODE REVIEW]

Possible answers:

A. {option_a}

B. {option_b}

C. {option_c}

D. {option_d}

Answer with the letter symbol only. Answer:

Figure 6: Few-Shot SI Prompt Template.

Pre-Review Code Submission

```
1    def main(args: argparse.Namespace):
2            )
3        host_environment = host_environments.pop()
4
5        module_dir_paths = sort_and_dedup_paths([
6            iree_artifacts .get_module_dir_path(config.module_generation_config)
7            for config in run_configs
8        ])
9
10       output_map[device_name] = {
11           "host_environment": dataclasses.asdict(host_environment),
```

Code Review: Huh, would be nice if the path was just naturally serializable

---

Which line numbers is the code review asking to modify code? A. ✓ line numbers 5, 6, 8

Change Localisation (Easy)

B. line numbers 1, 2, 3 C. line numbers 4, 7, 9 D. line numbers 9, 10, 11

Change Localisation (Hard)

B. line numbers 1, 5, 6 C. line numbers 3, 5, 6 D. line numbers 3, 5, 8

Which code revision is the code review asking for?

A. ✓

```
5 −  module_dir_paths = sort_and_dedup_paths([
5 +  module_dir_paths = sorted(set(
6 −      iree_artifacts .get_module_dir_path(config.module_generation_config)
6 +      str( iree_artifacts .get_module_dir_path(config.module_generation_config))
8 −  ])
8 +  ))
```

Solution Identification (Easy)

B.

```
5 −  module_dir_paths = sort_and_dedup_paths([
5 +  module_dir_paths = sorted(set(
6 −      iree_artifacts .get_module_dir_path(config.module_generation_config)
6 +      str( struct_lucule .get_module_dir_path(config.module_generation_config))
8 −  ])
8 +  ))
```

C.

```
5 −  module_dir_paths = sort_and_dedup_paths([
5 +  module_dir_paths = sorted(set(
6 −      iree_artifacts .get_module_dir_path(config.module_generation_config)
6 +      str( assertTrue_localhost .get_module_dir_path(config.module_generation_config))
8 −  ])
8 +  ))
```

D.

```
5 −  module_dir_paths = sort_and_dedup_paths([
5 +  module_dir_paths = sorted(set(
6 −      iree_artifacts .get_module_dir_path(config.module_generation_config)
6 +      str(indexChat_retry.get_module_dir_path(config.module_generation_config))
8 −  ])
8 +  ))
```

Solution Identification (Hard)

B.

```
5 −  module_dir_paths = sort_and_dedup_paths([
5 +  module_dir_paths = sorted(set(
6 −      iree_artifacts .get_module_dir_path(config.module_generation_config)
6 +      str(View_DEF.get_module_dir_path(config.module_generation_config))
8 −  ])
8 +  ))
```

C.

```
5 −  module_dir_paths = sort_and_dedup_paths([
5 +  module_dir_paths = sorted(set(
6 −      iree_artifacts .get_module_dir_path(config.module_generation_config)
6 +      str(develop_weight.get_module_dir_path(config.module_generation_config))
8 −  ])
8 +  ))
```

D.

```
5 −  module_dir_paths = sort_and_dedup_paths([
5 +  module_dir_paths = sorted(set(
6 −      iree_artifacts .get_module_dir_path(config.module_generation_config)
6 +      str( register_access .get_module_dir_path(config.module_generation_config))
8 −  ])
8 +  ))
```

Figure 7: Examples of Variation in Difficulty.

Comment Length = 8
FYI, this will spam console when running ‘aaa‘.

Comment Length = 18
By the format string it looks like parameters shall be reversed. Type shall be 1st and exception 2nd

---

Comment Length = 23
I’m wondering if it’s useful to show the message from the exception in this debug message, at least in the case of IOException.

Figure 8: Examples of Different Comment Lengths.

Code Edit Distance = 9

```
   OnConflictAction onconflict_action = ts_chunk_dispatch_get_on_conflict_action (dispatch);
   ResultRelInfo * resrelinfo , * relinfo ;
-  bool has_compressed_chunk = (chunk->fd.compressed_chunk_id != 0);
+  bool is_compressed = (chunk->fd.compressed_chunk_id != 0)
   /* permissions NOT checked here; were checked at hypertable level */
   if (check_enable_rls (chunk->table_id, InvalidOid , false ) == RLS_ENABLED)
```

Code Review: Should ‘is_compressed‘ now be given by the chunk’s compression status flag? Otherwise it looks like we have different ways of determining compression status.

Code Edit Distance = 25

```
   void Server_Card:: resetState ()
   setPT(QString());
   setAnnotation (QString());
   setDoesntUntap( false );
-    setFaceDown(false);
   }

   QString Server_Card:: setAttribute ( CardAttribute attribute , const QString &avalue, bool allCards )
```

Code Review: this causes a major bug: cards have their state reset when moved between the battlefield and the deck, their facedown state is then checked afterwards to determine what event to show to other players. with this change moving a facedown card to your deck (unknown to unknown) will tell all your opponents (but not you) what card it was.

Code Edit Distance = 67

```
    public unsafe LazyStringValue GetDocumentId(LazyStringValue key)
                if (index == -1)
                    return null;

-               _tmpLazyStringInstance = _context.GetLazyString(key.Buffer, index);
+               return _context.GetLazyString(key.Buffer, index);
-               return _tmpLazyStringInstance;
        }

        // TODO unify if possible with AllowedPathsValidator
```

Code Review: Why do you store that in the temporary variable?

Figure 9: Examples of Different Code Edit Distances.

Code Element Ratio = 0
Should this really be a compile time error? The fact that is can be imported multiple times does not mean that it will be.

Code Element Ratio = 0.13
For all of the fuzz tests, does it make sense to have versions for ‘len_prefixed‘ both ‘true‘ and ‘false‘ ?

Code Element Ratio = 0.37
I think you’re missing a ‘flb_free(seq_index_str);‘ there.
Other than that, would you mind change that comparison to ‘if (tmp_key == NULL) {‘ instead? I’d really appreciate it.

Figure 10: Examples of Different Code Element Ratios.

Specification Ratio = 0.67

```
  void hpx_thread_buffer :: resize (const std :: size_t num_threads,
  }

  void *hpx_thread_buffer :: get (std :: size_t thread_num) const noexcept {
-     KOKKOS_ASSERT(thread_num < m_num_threads);
+     KOKKOS_EXPECTS(thread_num < m_num_threads);
      if (m_data == nullptr ) {
          return nullptr ;
      }
      return &m_data[thread_num * m_size_per_thread];
   }

  void *hpx_thread_buffer :: get_extra_space () const noexcept {
-     KOKKOS_ASSERT(m_extra_space > 0);
+     KOKKOS_EXPECTS(m_extra_space > 0);
      if (m_data == nullptr ) {
          return nullptr ;
      }
```

Code Review: This is fine but just pointing out there is also a 'KOKKOS_EXPECTS' that was meant for checking preconditions

---

Specification Ratio = 37.60

```
  private String addNashornJavaScriptEngineIfNecessary( String cp) {
      }

      private boolean requiresNashornJavaScriptEngine () {
-         String version = System.getProperty ("java. specification . version");
+         return getJavaVersion () >= 15; // Nashorn was removed in Java 15
-         if (version . startsWith ("1.")) {
-             version = version . substring (2);
-         }
-         return Integer . parseInt ( version ) >= 15; // Nashorn was removed in Java 15
      }

  }
```

Code Review: You can use 'getJavaVersion()' here.

Figure 11: Examples of Different Specification Ratios.

Scale	Model	Parameters	Organisation	Release Date	Hugging Face
≤3B	Llama-3.2-3B-Instruct	3.2B	Meta	Sep 25, 2024	meta-llama/Llama-3.2-3B-Instruct
	Llama-3.2-1B-Instruct	1.2B	Meta	Sep 25, 2024	meta-llama/Llama-3.2-1B-Instruct
	Qwen2.5-Coder-3B-Instruct	3.1B	Alibaba	Nov 6, 2024	Qwen/Qwen2.5-Coder-3B-Instruct
	Qwen2.5-Coder-1.5B-Instruct	1.5B	Alibaba	Sep 18, 2024	Qwen/Qwen2.5-Coder-1.5B-Instruct
	Qwen2.5-3B-Instruct	3.1B	Alibaba	Sep 18, 2024	Qwen/Qwen2.5-3B-Instruct
	Qwen2.5-1.5B-Instruct	1.5B	Alibaba	Sep 18, 2024	Qwen/Qwen2.5-1.5B-Instruct
	deepseek-coder-1.3b-instruct	1.3B	DeepSeek	Oct 30, 2023	deepseek-ai/deepseek-coder-1.3b-instruct
	DeepSeek-R1-Distill-Qwen-1.5B	1.5B	DeepSeek	Jan 20, 2025	deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B
	Falcon3-3B-Instruct	3.2B	TII UAE	Dec 17, 2024	tiiuae/Falcon3-3B-Instruct
	Falcon3-1B-Instruct	1.7B	TII UAE	Dec 17, 2024	tiiuae/Falcon3-1B-Instruct
	Phi-3-mini-128k-instruct	3.8B	Microsoft	Apr 23, 2024	microsoft/Phi-3-mini-128k-instruct
	Yi-Coder-1.5B-Chat	1.5B	01.AI	Aug 27, 2024	01-ai/Yi-Coder-1.5B-Chat
	granite-3b-code-instruct-128k	3.5B	IBM	Jul 18, 2024	ibm-granite/granite-3b-code-instruct-128k
	granite-3.0-3b-a800m-instruct	3.4B	IBM	Oct 21, 2024	ibm-granite/granite-3.0-3b-a800m-instruct
	granite-3.0-2b-instruct	2.6B	IBM	Oct 21, 2024	ibm-granite/granite-3.0-2b-instruct
	EXAONE-3.5-2.4B-Instruct	2.4B	LG AI	Dec 9, 2024	LGAI-EXAONE/EXAONE-3.5-2.4B-Instruct
	internlm2_5-1_8b-chat	1.9B	InternLM	Jul 30, 2024	internlm/internlm2_5-1_8b-chat
	stable-code-instruct-3b	2.8B	Stability AI	Mar 19, 2024	stabilityai/stable-code-instruct-3b
≤9B	CodeLlama-7b-Instruct-hf	6.7B	Meta	Mar 13, 2024	meta-llama/CodeLlama-7b-Instruct-hf
	Llama-3.1-8B-Instruct	8.0B	Meta	Jul 18, 2024	meta-llama/Llama-3.1-8B-Instruct
	codegemma-1.1-7b-it	8.5B	Google	Apr 30, 2024	google/codegemma-1.1-7b-it
	gemma-2-9b-it	9.2B	Google	Jun 25, 2024	google/gemma-2-9b-it
	Qwen2.5-Coder-7B-Instruct	7.6B	Alibaba	Sep 17, 2024	Qwen/Qwen2.5-Coder-7B-Instruct
	Qwen2.5-7B-Instruct	7.6B	Alibaba	Sep 16, 2024	Qwen/Qwen2.5-7B-Instruct
	Marco-o1	7.6B	AIDC-AI	Nov 13, 2024	AIDC-AI/Marco-o1
	deepseek-coder-7b-instruct-v1.5	6.9B	DeepSeek	Jan 25, 2024	deepseek-ai/deepseek-coder-7b-instruct-v1.5
	deepseek-llm-7b-chat	6.9B	DeepSeek	Nov 29, 2023	deepseek-ai/deepseek-llm-7b-chat
	DeepSeek-R1-Distill-Qwen-7B	7.6B	DeepSeek	Jan 20, 2025	deepseek-ai/DeepSeek-R1-Distill-Qwen-7B
	DeepSeek-R1-Distill-Llama-8B	8.0B	DeepSeek	Jan 20, 2025	deepseek-ai/DeepSeek-R1-Distill-Llama-8B
	Falcon3-7B-Instruct	7.5B	TII UAE	Dec 17, 2024	tiiuae/Falcon3-7B-Instruct
	Baichuan2-7B-Chat	7.1B	Baichuan AI	Sep 6, 2023	baichuan-inc/Baichuan2-7B-Chat
	Yi-Coder-9B-Chat	8.8B	01.AI	Aug 27, 2024	01-ai/Yi-Coder-9B-Chat
	Yi-1.5-9B-Chat	8.8B	01.AI	May 10, 2024	01-ai/Yi-1.5-9B-Chat
	granite-8b-code-instruct-128k	8.1B	IBM	Jul 12, 2024	ibm-granite/granite-8b-code-instruct-128k
	granite-3.0-8b-instruct	8.2B	IBM	Oct 15, 2024	ibm-granite/granite-3.0-8b-instruct
	EXAONE-3.5-7.8B-Instruct	7.8B	LG AI	Dec 9, 2024	LGAI-EXAONE/EXAONE-3.5-7.8B-Instruct
≤16B	CodeLlama-13b-Instruct-hf	13.0B	Meta	Mar 13, 2024	meta-llama/CodeLlama-13b-Instruct-hf
	Qwen2.5-Coder-14B-Instruct	14.8B	Alibaba	Nov 6, 2024	Qwen/Qwen2.5-Coder-14B-Instruct
	Qwen2.5-14B-Instruct	14.8B	Alibaba	Sep 16, 2024	Qwen/Qwen2.5-14B-Instruct
	DeepSeek-Coder-V2-Lite-Instruct	15.7B	DeepSeek	Jun 14, 2024	deepseek-ai/DeepSeek-Coder-V2-Lite-Instruct
	DeepSeek-V2-Lite-Chat	15.7B	DeepSeek	May 15, 2024	deepseek-ai/DeepSeek-V2-Lite-Chat
	DeepSeek-R1-Distill-Qwen-14B	14.8B	DeepSeek	Jan 20, 2025	deepseek-ai/DeepSeek-R1-Distill-Qwen-14B
	falcon-11B	11.1B	TII UAE	May 9, 2024	tiiuae/falcon-11B
	Falcon3-10B-Instruct	10.3B	TII UAE	Dec 17, 2024	tiiuae/Falcon3-10B-Instruct
	Baichuan2-13B-Chat	13.0B	Baichuan AI	Sep 6, 2023	baichuan-inc/Baichuan2-13B-Chat
	WizardLM-13B-V1.2	13.0B	WizardLM Team	Jul 25, 2023	WizardLMTeam/WizardLM-13B-V1.2
	Phi-3-medium-128k-instruct	14.7B	Microsoft	May 2, 2024	microsoft/Phi-3-medium-128k-instruct
	phi-4	14.7B	Microsoft	Dec 12, 2024	microsoft/phi-4
	starcoder2-15b-instruct-v0.1	16.0B	BigCode	Apr 23, 2024	bigcode/starcoder2-15b-instruct-v0.1
	Mistral-Nemo-Instruct-2407	12.2B	Mistral	Jul 17, 2024	mistralai/Mistral-Nemo-Instruct-2407
≤34B	CodeLlama-34b-Instruct-hf	33.7B	Meta	Mar 14, 2024	meta-llama/CodeLlama-34b-Instruct-hf
	gemma-2-27b-it	27.2B	Google	Jun 25, 2024	google/gemma-2-27b-it
	Qwen2.5-Coder-32B-Instruct	32.8B	Alibaba	Nov 6, 2024	Qwen/Qwen2.5-Coder-32B-Instruct
	Qwen2.5-32B-Instruct	32.8B	Alibaba	Sep 17, 2024	Qwen/Qwen2.5-32B-Instruct
	QwQ-32B	32.8B	Alibaba	Mar 6, 2025	Qwen/QwQ-32B
	Sky-T1-32B-Preview	32.8B	NovaSky	Jan 9, 2025	NovaSky-AI/Sky-T1-32B-Preview
	deepseek-coder-33b-instruct	33.3B	DeepSeek	Nov 1, 2023	deepseek-ai/deepseek-coder-33b-instruct
	DeepSeek-R1-Distill-Qwen-32B	32.8B	DeepSeek	Jan 20, 2025	deepseek-ai/DeepSeek-R1-Distill-Qwen-32B
	Yi-1.5-34B-Chat	34.4B	01.AI	May 10, 2024	01-ai/Yi-1.5-34B-Chat
	Mistral-Small-Instruct-2409	22.2B	Mistral	Sep 17, 2024	mistralai/Mistral-Small-Instruct-2409
	granite-34b-code-instruct-8k	33.7B	IBM	May 4, 2024	ibm-granite/granite-34b-code-instruct-8k
	internlm2_5-20b-chat	19.9B	InternLM	Jul 30, 2024	internlm/internlm2_5-20b-chat
	EXAONE-3.5-32B-Instruct	32.0B	LG AI	Dec 9, 2024	LGAI-EXAONE/EXAONE-3.5-32B-Instruct
≤72B	CodeLlama-70b-Instruct-hf	69.0B	Meta	Mar 14, 2024	meta-llama/CodeLlama-70b-Instruct-hf
	Llama-3.1-70B-Instruct	70.6B	Meta	Jul 16, 2024	meta-llama/Llama-3.1-70B-Instruct
	Llama-3.3-70B-Instruct	70.6B	Meta	Nov 26, 2024	meta-llama/Llama-3.3-70B-Instruct
	Qwen2.5-72B-Instruct	72.7B	Alibaba	Sep 16, 2024	Qwen/Qwen2.5-72B-Instruct
	deepseek-llm-67b-chat	67.0B	DeepSeek	Nov 29, 2023	deepseek-ai/deepseek-llm-67b-chat
	DeepSeek-R1-Distill-Llama-70B	70.6B	DeepSeek	Jan 20, 2025	deepseek-ai/DeepSeek-R1-Distill-Llama-70B
	WizardLM-70B-V1.0	70.0B	WizardLM Team	Aug 9, 2023	WizardLMTeam/WizardLM-70B-V1.0
	K2-Chat	65.3B	LLM360	May 28, 2024	LLM360/K2-Chat
	falcon-40b-instruct	40.0B	TII UAE	May 25, 2023	tiiuae/falcon-40b-instruct

Table 7: List of benchmarked models.

Scale	Model	C	C++	CSharp	Go	Java	JavaScript	PHP	Python	Ruby	Overall
≤3B	Llama-3.2-3B-Instruct	23.0	18.0	22.0	21.0	29.0	24.0	36.0	33.0	27.0	25.9
	Llama-3.2-1B-Instruct	4.0	7.0	1.0	9.0	11.0	2.0	9.0	5.0	3.0	5.7
	Qwen2.5-Coder-3B-Instruct	24.0	28.0	27.0	30.0	30.0	25.0	42.0	30.0	37.0	30.3
	Qwen2.5-Coder-1.5B-Instruct	11.0	16.0	19.0	23.0	23.0	19.0	36.0	23.0	21.0	21.2
	Qwen2.5-3B-Instruct	15.0	16.0	12.0	24.0	18.0	26.0	31.0	24.0	26.0	21.3
	Qwen2.5-1.5B-Instruct	14.0	17.0	12.0	16.0	24.0	19.0	27.0	23.0	28.0	20.0
	deepseek-coder-1.3b-instruct	7.0	8.0	10.0	13.0	13.0	6.0	16.0	16.0	16.0	11.7
	DeepSeek-R1-Distill-Qwen-1.5B	0.0	5.0	3.0	4.0	1.0	2.0	2.0	2.0	1.0	2.2
	Falcon3-3B-Instruct	11.0	14.0	7.0	19.0	14.0	16.0	21.0	8.0	18.0	14.2
	Falcon3-1B-Instruct	2.0	3.0	2.0	7.0	5.0	4.0	4.0	2.0	2.0	3.4
	Phi-3-mini-128k-instruct	15.0	6.0	3.0	2.0	1.0	11.0	8.0	8.0	15.0	7.7
	Yi-Coder-1.5B-Chat	5.0	6.0	3.0	4.0	4.0	4.0	10.0	5.0	10.0	5.7
	granite-3b-code-instruct-128k	21.0	23.0	21.0	28.0	27.0	23.0	24.0	30.0	24.0	24.6
	granite-3.0-3b-a800m-instruct	10.0	14.0	6.0	13.0	21.0	11.0	18.0	23.0	16.0	14.7
	granite-3.0-2b-instruct	16.0	18.0	13.0	16.0	19.0	19.0	29.0	29.0	25.0	20.4
	EXAONE-3.5-2.4B-Instruct	5.0	6.0	1.0	0.0	2.0	4.0	4.0	3.0	9.0	3.8
	internlm2_5-1_8b-chat	5.0	5.0	2.0	3.0	6.0	4.0	7.0	7.0	4.0	4.8
	stable-code-instruct-3b	8.0	4.0	1.0	4.0	2.0	2.0	10.0	10.0	5.0	5.1
≤9B	CodeLlama-7b-Instruct-hf	30.0	27.0	27.0	36.0	41.0	30.0	43.0	40.0	41.0	35.0
	Llama-3.1-8B-Instruct	28.0	31.0	16.0	17.0	16.0	20.0	30.0	38.0	40.0	26.2
	codegemma-1.1-7b-it	22.0	10.0	25.0	32.0	21.0	32.0	26.0	27.0	41.0	26.2
	gemma-2-9b-it	29.0	31.0	44.0	39.0	34.0	39.0	46.0	47.0	42.0	39.0
	Qwen2.5-Coder-7B-Instruct	29.0	38.0	44.0	37.0	40.0	39.0	47.0	47.0	48.0	41.0
	Qwen2.5-7B-Instruct	22.0	21.0	33.0	34.0	41.0	36.0	43.0	43.0	48.0	35.7
	Marco-o1	27.0	28.0	33.0	37.0	34.0	32.0	40.0	41.0	43.0	35.0
	deepseek-coder-7b-instruct-v1.5	33.0	34.0	28.0	39.0	37.0	34.0	36.0	44.0	42.0	36.3
	deepseek-llm-7b-chat	15.0	15.0	14.0	17.0	21.0	14.0	20.0	26.0	18.0	17.8
	DeepSeek-R1-Distill-Qwen-7B	3.0	10.0	6.0	8.0	9.0	7.0	4.0	4.0	9.0	6.7
	DeepSeek-R1-Distill-Llama-8B	3.0	5.0	3.0	6.0	1.0	8.0	3.0	6.0	9.0	4.9
	Falcon3-7B-Instruct	16.0	15.0	7.0	14.0	12.0	14.0	19.0	11.0	21.0	14.3
	Baichuan2-7B-Chat	9.0	15.0	8.0	15.0	18.0	11.0	21.0	18.0	13.0	14.2
	Yi-Coder-9B-Chat	23.0	27.0	20.0	35.0	37.0	28.0	40.0	42.0	40.0	32.4
	Yi-1.5-9B-Chat	24.0	30.0	22.0	30.0	29.0	34.0	31.0	38.0	38.0	30.7
	granite-8b-code-instruct-128k	22.0	28.0	25.0	34.0	38.0	30.0	41.0	41.0	43.0	33.6
	granite-3.0-8b-instruct	23.0	24.0	30.0	29.0	31.0	27.0	43.0	37.0	42.0	31.8
	EXAONE-3.5-7.8B-Instruct	15.0	12.0	19.0	20.0	12.0	8.0	22.0	20.0	26.0	17.1
≤16B	CodeLlama-13b-Instruct-hf	29.0	32.0	30.0	37.0	43.0	36.0	43.0	42.0	38.0	36.7
	Qwen2.5-Coder-14B-Instruct	36.0	34.0	**45.0**	43.0	47.0	46.0	54.0	54.0	60.0	46.6
	Qwen2.5-14B-Instruct	26.0	31.0	29.0	34.0	30.0	38.0	49.0	42.0	52.0	36.8
	DeepSeek-Coder-V2-Lite-Instruct	20.0	29.0	10.0	23.0	33.0	25.0	38.0	30.0	35.0	27.0
	DeepSeek-V2-Lite-Chat	12.0	7.0	10.0	14.0	24.0	16.0	16.0	22.0	14.0	15.0
	DeepSeek-R1-Distill-Qwen-14B	25.0	27.0	20.0	32.0	28.0	29.0	46.0	41.0	34.0	31.3
	falcon-11B	16.0	14.0	22.0	23.0	19.0	17.0	23.0	23.0	20.0	19.7
	Falcon3-10B-Instruct	19.0	23.0	14.0	14.0	24.0	15.0	26.0	27.0	39.0	22.3
	Baichuan2-13B-Chat	11.0	14.0	12.0	17.0	10.0	13.0	24.0	25.0	19.0	16.1
	WizardLM-13B-V1.2	15.0	20.0	21.0	20.0	19.0	23.0	32.0	31.0	25.0	22.9
	Phi-3-medium-128k-instruct	21.0	22.0	24.0	32.0	35.0	29.0	35.0	38.0	36.0	30.2
	phi-4	41.0	23.0	38.0	36.0	38.0	37.0	37.0	48.0	36.0	37.1
	starcoder2-15b-instruct-v0.1	25.0	34.0	29.0	40.0	39.0	29.0	29.0	39.0	39.0	33.7
	Mistral-Nemo-Instruct-2407	29.0	22.0	27.0	33.0	29.0	38.0	38.0	39.0	43.0	33.1
≤34B	CodeLlama-34b-Instruct-hf	37.0	36.0	38.0	47.0	45.0	41.0	51.0	46.0	47.0	43.1
	gemma-2-27b-it	38.0	37.0	42.0	47.0	43.0	47.0	**56.0**	54.0	54.0	46.4
	Qwen2.5-Coder-32B-Instruct	35.0	35.0	40.0	45.0	40.0	39.0	51.0	51.0	61.0	44.1
	Qwen2.5-32B-Instruct	38.0	41.0	40.0	**52.0**	**49.0**	32.0	51.0	47.0	55.0	45.0
	QwQ-32B	25.0	26.0	31.0	34.0	38.0	26.0	45.0	36.0	35.0	32.9
	Sky-T1-32B-Preview	36.0	38.0	35.0	49.0	43.0	32.0	52.0	46.0	51.0	42.4
	deepseek-coder-33b-instruct	28.0	30.0	22.0	40.0	32.0	28.0	38.0	40.0	37.0	32.8
	DeepSeek-R1-Distill-Qwen-32B	25.0	23.0	36.0	22.0	25.0	31.0	36.0	35.0	42.0	30.6
	Yi-1.5-34B-Chat	22.0	30.0	28.0	28.0	27.0	27.0	39.0	39.0	40.0	31.1
	Mistral-Small-Instruct-2409	34.0	39.0	44.0	42.0	42.0	51.0	53.0	50.0	55.0	45.6
	granite-34b-code-instruct-8k	23.0	27.0	31.0	36.0	35.0	32.0	48.0	45.0	42.0	35.4
	internlm2_5-20b-chat	29.0	24.0	31.0	37.0	33.0	28.0	40.0	40.0	39.0	33.4
	EXAONE-3.5-32B-Instruct	27.0	28.0	28.0	24.0	28.0	29.0	39.0	32.0	39.0	30.4
≤72B	CodeLlama-70b-Instruct-hf	42.0	38.0	38.0	48.0	48.0	41.0	54.0	53.0	45.0	45.2
	Llama-3.1-70B-Instruct	**46.0**	42.0	**45.0**	49.0	47.0	**57.0**	55.0	50.0	**62.0**	**50.3**
	Llama-3.3-70B-Instruct	40.0	33.0	36.0	19.0	31.0	34.0	40.0	17.0	46.0	32.9
	Qwen2.5-72B-Instruct	**46.0**	**45.0**	44.0	44.0	48.0	44.0	53.0	**56.0**	58.0	48.7
	deepseek-llm-67b-chat	29.0	33.0	31.0	41.0	35.0	43.0	50.0	44.0	51.0	39.7
	DeepSeek-R1-Distill-Llama-70B	20.0	23.0	23.0	23.0	26.0	43.0	35.0	24.0	41.0	28.7
	WizardLM-70B-V1.0	16.0	31.0	29.0	28.0	32.0	33.0	36.0	32.0	41.0	30.9
	K2-Chat	34.0	35.0	36.0	41.0	35.0	33.0	50.0	48.0	47.0	39.9
	falcon-40b-instruct	17.0	13.0	12.0	20.0	21.0	21.0	27.0	24.0	16.0	19.0

Best Score Overall, Best Score within Scale

Table 8: Automated Code Refinement - Exact Match Rate (%).

Scale	Model	PPA (Random = 33.3)	C	C++	CSharp	Go	Java	JavaScript	PHP	Python	Ruby	Overall
≤3B	Llama-3.2-3B-Instruct	99.2 (+65.8)	<u>73.0</u>	<u>80.0</u>	73.0	<u>80.0</u>	82.0	79.0	<u>85.0</u>	72.0	85.0	<u>78.8</u>
	Llama-3.2-1B-Instruct	98.3 (+65.0)	72.0	76.0	72.0	77.0	79.0	77.0	80.0	73.0	79.0	76.1
	Qwen2.5-Coder-3B-Instruct	98.6 (+65.3)	72.0	76.0	72.0	<u>80.0</u>	81.0	<u>80.0</u>	81.0	71.0	86.0	77.7
	Qwen2.5-Coder-1.5B-Instruct	95.7 (+62.3)	67.0	74.0	67.0	75.0	76.0	73.0	76.0	65.0	81.0	72.7
	Qwen2.5-3B-Instruct	94.0 (+60.6)	70.0	76.0	64.0	72.0	72.0	64.0	68.0	59.0	74.0	68.8
	Qwen2.5-1.5B-Instruct	97.7 (+64.4)	71.0	79.0	70.0	<u>80.0</u>	<u>83.0</u>	78.0	82.0	<u>74.0</u>	<u>**87.0**</u>	78.2
	deepseek-coder-1.3b-instruct	43.9 (+10.6)	11.0	0.0	1.0	1.0	3.0	0.0	0.0	0.0	1.0	1.9
	DeepSeek-R1-Distill-Qwen-1.5B	36.8 (+3.4)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Falcon3-3B-Instruct	86.0 (+52.6)	42.0	58.0	50.0	53.0	53.0	56.0	59.0	43.0	54.0	52.0
	Falcon3-1B-Instruct	89.2 (+55.9)	36.0	64.0	33.0	66.0	62.0	52.0	66.0	53.0	70.0	55.8
	Phi-3-mini-128k-instruct	96.9 (+63.5)	<u>73.0</u>	77.0	<u>74.0</u>	74.0	81.0	77.0	81.0	72.0	83.0	76.9
	Yi-Coder-1.5B-Chat	62.1 (+28.7)	0.0	5.0	5.0	25.0	3.0	2.0	5.0	7.0	9.0	6.8
	granite-3b-code-instruct-128k	80.4 (+47.0)	28.0	23.0	26.0	40.0	36.0	30.0	21.0	22.0	26.0	28.0
	granite-3.0-3b-a800m-instruct	82.9 (+49.5)	50.0	37.0	28.0	37.0	17.0	19.0	40.0	23.0	17.0	29.8
	granite-3.0-2b-instruct	94.2 (+60.9)	66.0	69.0	69.0	70.0	76.0	69.0	72.0	54.0	76.0	69.0
	EXAONE-3.5-2.4B-Instruct	92.7 (+59.4)	55.0	71.0	65.0	67.0	69.0	58.0	73.0	52.0	78.0	65.3
	internlm2_5-1_8b-chat	95.3 (+62.0)	70.0	69.0	58.0	68.0	67.0	52.0	79.0	60.0	57.0	64.4
	stable-code-instruct-3b	51.1 (+17.7)	0.0	0.0	0.0	0.0	3.0	2.0	1.0	0.0	2.0	0.9
≤9B	CodeLlama-7b-Instruct-hf	97.6 (+64.3)	68.0	78.0	73.0	78.0	78.0	75.0	81.0	66.0	82.0	75.4
	Llama-3.1-8B-Instruct	96.0 (+62.7)	68.0	74.0	65.0	77.0	72.0	74.0	77.0	62.0	77.0	71.8
	codegemma-1.1-7b-it	99.6 (+66.3)	<u>73.0</u>	80.0	75.0	81.0	<u>83.0</u>	<u>80.0</u>	<u>**86.0**</u>	74.0	<u>**87.0**</u>	<u>**79.9**</u>
	gemma-2-9b-it	94.8 (+61.5)	72.0	80.0	69.0	74.0	78.0	75.0	76.0	59.0	84.0	74.1
	Qwen2.5-Coder-7B-Instruct	97.7 (+64.4)	71.0	80.0	73.0	80.0	<u>83.0</u>	<u>80.0</u>	85.0	70.0	85.0	78.6
	Qwen2.5-7B-Instruct	95.9 (+62.6)	72.0	79.0	74.0	79.0	81.0	74.0	70.0	71.0	83.0	75.9
	Marco-o1	97.3 (+63.9)	<u>73.0</u>	<u>81.0</u>	<u>76.0</u>	81.0	82.0	74.0	76.0	<u>**75.0**</u>	85.0	78.1
	deepseek-coder-7b-instruct-v1.5	89.8 (+56.4)	58.0	68.0	58.0	65.0	54.0	51.0	61.0	44.0	64.0	58.1
	deepseek-llm-7b-chat	97.6 (+64.2)	66.0	76.0	66.0	75.0	78.0	74.0	80.0	66.0	79.0	73.3
	DeepSeek-R1-Distill-Qwen-7B	39.6 (+6.3)	1.0	1.0	0.0	0.0	1.0	2.0	0.0	1.0	3.0	1.0
	DeepSeek-R1-Distill-Llama-8B	50.6 (+17.3)	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	1.0	0.4
	Falcon3-7B-Instruct	94.7 (+61.4)	66.0	76.0	73.0	75.0	79.0	73.0	74.0	67.0	76.0	73.2
	Baichuan2-7B-Chat	95.6 (+62.3)	25.0	77.0	74.0	80.0	77.0	78.0	84.0	68.0	82.0	71.7
	Yi-Coder-9B-Chat	87.5 (+54.1)	57.0	66.0	44.0	54.0	50.0	50.0	63.0	39.0	46.0	52.1
	Yi-1.5-9B-Chat	97.7 (+64.4)	<u>73.0</u>	80.0	75.0	<u>**83.0**</u>	<u>83.0</u>	77.0	79.0	72.0	85.0	78.6
	granite-8b-code-instruct-128k	99.0 (+65.7)	71.0	78.0	71.0	77.0	77.0	76.0	81.0	69.0	82.0	75.8
	granite-3.0-8b-instruct	88.8 (+55.4)	56.0	62.0	47.0	57.0	51.0	53.0	68.0	45.0	64.0	55.9
	EXAONE-3.5-7.8B-Instruct	94.1 (+60.8)	67.0	66.0	63.0	70.0	65.0	66.0	74.0	60.0	77.0	67.6
≤16B	CodeLlama-13b-Instruct-hf	96.0 (+62.7)	42.0	78.0	68.0	68.0	66.0	74.0	77.0	58.0	79.0	67.8
	Qwen2.5-Coder-14B-Instruct	95.8 (+62.5)	69.0	79.0	66.0	74.0	77.0	74.0	82.0	64.0	80.0	73.9
	Qwen2.5-14B-Instruct	94.9 (+61.6)	68.0	73.0	76.0	74.0	80.0	73.0	77.0	62.0	77.0	73.3
	DeepSeek-Coder-V2-Lite-Instruct	96.1 (+62.8)	70.0	<u>81.0</u>	<u>77.0</u>	80.0	79.0	69.0	75.0	72.0	78.0	75.7
	DeepSeek-V2-Lite-Chat	97.9 (+64.5)	72.0	80.0	66.0	79.0	<u>83.0</u>	77.0	84.0	69.0	81.0	76.8
	DeepSeek-R1-Distill-Qwen-14B	95.6 (+62.3)	67.0	68.0	72.0	72.0	77.0	76.0	78.0	64.0	80.0	72.7
	falcon-11B	99.7 (+66.4)	72.0	80.0	74.0	<u>81.0</u>	<u>83.0</u>	<u>80.0</u>	<u>85.0</u>	<u>74.0</u>	<u>**87.0**</u>	<u>79.6</u>
	Falcon3-10B-Instruct	86.3 (+53.0)	71.0	54.0	53.0	44.0	50.0	26.0	51.0	31.0	22.0	44.7
	Baichuan2-13B-Chat	93.8 (+60.4)	14.0	80.0	67.0	76.0	78.0	75.0	83.0	69.0	76.0	68.7
	WizardLM-13B-V1.2	94.4 (+61.1)	42.0	69.0	66.0	68.0	70.0	51.0	74.0	67.0	79.0	65.1
	Phi-3-medium-128k-instruct	97.3 (+63.9)	71.0	78.0	75.0	79.0	77.0	76.0	81.0	68.0	85.0	76.7
	phi-4	96.6 (+63.3)	<u>73.0</u>	77.0	74.0	78.0	80.0	76.0	78.0	69.0	84.0	76.6
	starcoder2-15b-instruct-v0.1	88.2 (+54.9)	48.0	43.0	31.0	42.0	38.0	43.0	50.0	37.0	29.0	40.1
	Mistral-Nemo-Instruct-2407	96.1 (+62.7)	69.0	78.0	73.0	80.0	76.0	76.0	78.0	69.0	83.0	75.8
≤34B	CodeLlama-34b-Instruct-hf	50.2 (+16.9)	1.0	4.0	1.0	0.0	1.0	0.0	2.0	2.0	3.0	1.6
	gemma-2-27b-it	94.8 (+61.5)	69.0	77.0	70.0	76.0	73.0	73.0	77.0	68.0	83.0	74.0
	Qwen2.5-Coder-32B-Instruct	97.0 (+63.7)	69.0	82.0	<u>81.0</u>	<u>82.0</u>	<u>**85.0**</u>	77.0	80.0	<u>72.0</u>	84.0	<u>79.1</u>
	Qwen2.5-32B-Instruct	93.1 (+59.7)	<u>73.0</u>	76.0	71.0	68.0	78.0	72.0	64.0	67.0	70.0	71.0
	QwQ-32B	89.8 (+56.5)	55.0	62.0	63.0	58.0	69.0	67.0	59.0	54.0	61.0	60.9
	Sky-T1-32B-Preview	89.9 (+56.6)	56.0	68.0	61.0	58.0	71.0	64.0	60.0	61.0	64.0	62.6
	deepseek-coder-33b-instruct	96.0 (+62.7)	42.0	70.0	60.0	76.0	74.0	72.0	78.0	71.0	85.0	69.8
	DeepSeek-R1-Distill-Qwen-32B	93.8 (+60.5)	69.0	77.0	76.0	74.0	80.0	72.0	72.0	63.0	76.0	73.2
	Yi-1.5-34B-Chat	93.3 (+60.0)	60.0	67.0	58.0	69.0	71.0	67.0	74.0	56.0	73.0	66.1
	Mistral-Small-Instruct-2409	96.3 (+62.9)	72.0	82.0	76.0	79.0	78.0	75.0	79.0	69.0	80.0	76.7
	granite-34b-code-instruct-8k	96.5 (+63.1)	66.0	78.0	71.0	79.0	78.0	65.0	83.0	57.0	74.0	72.3
	internlm2_5-20b-chat	97.5 (+64.1)	72.0	77.0	67.0	80.0	78.0	75.0	<u>84.0</u>	67.0	81.0	75.7
	EXAONE-3.5-32B-Instruct	96.7 (+63.4)	71.0	<u>**83.0**</u>	74.0	81.0	82.0	<u>78.0</u>	81.0	66.0	<u>86.0</u>	78.0
≤72B	CodeLlama-70b-Instruct-hf	98.1 (+64.7)	72.0	79.0	71.0	81.0	<u>83.0</u>	75.0	83.0	73.0	83.0	77.8
	Llama-3.1-70B-Instruct	92.6 (+59.3)	59.0	71.0	75.0	68.0	71.0	74.0	67.0	63.0	68.0	68.4
	Llama-3.3-70B-Instruct	91.9 (+58.5)	59.0	68.0	69.0	67.0	70.0	73.0	65.0	59.0	64.0	66.0
	Qwen2.5-72B-Instruct	97.1 (+63.8)	<u>**75.0**</u>	81.0	<u>**82.0**</u>	80.0	<u>83.0</u>	79.0	77.0	<u>**75.0**</u>	<u>86.0</u>	<u>79.8</u>
	deepseek-llm-67b-chat	98.1 (+64.8)	74.0	81.0	73.0	81.0	82.0	79.0	<u>84.0</u>	72.0	<u>86.0</u>	79.1
	DeepSeek-R1-Distill-Llama-70B	90.1 (+56.8)	50.0	68.0	63.0	55.0	67.0	65.0	60.0	60.0	63.0	61.2
	WizardLM-70B-V1.0	97.9 (+64.6)	72.0	<u>82.0</u>	73.0	80.0	80.0	<u>**81.0**</u>	82.0	73.0	84.0	78.6
	K2-Chat	97.4 (+64.1)	72.0	79.0	71.0	<u>82.0</u>	<u>83.0</u>	77.0	81.0	72.0	<u>86.0</u>	78.1
	falcon-40b-instruct	92.0 (+58.6)	12.0	57.0	58.0	72.0	69.0	65.0	74.0	57.0	77.0	60.1

PPA: Proportion of Plurarity Agreement, **Best Score Overall**, <u>Best Score within Scale</u>

Table 9: Change Type Recognition - Proportion of Plurarity Agreement and Invariant Accuracy (%).

Scale	Model	PPA (Random = 25.0)	C	C++	CSharp	Go	Java	JavaScript	PHP	Python	Ruby	Overall
≤3B	Llama-3.2-3B-Instruct	47.3 (+22.3)	2.0	0.0	0.0	0.0	1.0	0.0	2.0	1.0	1.0	0.8
	Llama-3.2-1B-Instruct	32.9 (+7.9)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Qwen2.5-Coder-3B-Instruct	49.6 (+24.6)	0.0	4.0	1.0	2.0	2.0	0.0	2.0	2.0	3.0	1.8
	Qwen2.5-Coder-1.5B-Instruct	25.7 (+0.7)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Qwen2.5-3B-Instruct	82.4 (+57.4)	41.0	34.0	45.0	39.0	45.0	31.0	46.0	37.0	36.0	39.3
	Qwen2.5-1.5B-Instruct	57.6 (+32.6)	2.0	1.0	2.0	1.0	3.0	1.0	4.0	1.0	1.0	1.8
	deepseek-coder-1.3b-instruct	25.2 (+0.2)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	DeepSeek-R1-Distill-Qwen-1.5B	25.0 (+0.0)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Falcon3-3B-Instruct	59.9 (+34.9)	2.0	3.0	4.0	4.0	6.0	2.0	2.0	0.0	1.0	2.7
	Falcon3-1B-Instruct	25.1 (+0.1)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Phi-3-mini-128k-instruct	78.9 (+53.9)	27.0	24.0	34.0	38.0	32.0	28.0	48.0	43.0	33.0	34.1
	Yi-Coder-1.5B-Chat	35.5 (+10.5)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	granite-3b-code-instruct-128k	42.9 (+17.9)	0.0	1.0	0.0	1.0	1.0	1.0	2.0	1.0	0.0	0.8
	granite-3.0-3b-a800m-instruct	48.6 (+23.6)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	granite-3.0-2b-instruct	50.9 (+25.9)	4.0	2.0	3.0	1.0	1.0	1.0	4.0	0.0	0.0	1.8
	EXAONE-3.5-2.4B-Instruct	45.1 (+20.1)	3.0	1.0	1.0	0.0	0.0	5.0	1.0	2.0	0.0	1.4
	internlm2_5-1_8b-chat	49.1 (+24.1)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	stable-code-instruct-3b	38.5 (+13.5)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
≤9B	CodeLlama-7b-Instruct-hf	49.1 (+24.1)	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.3
	Llama-3.1-8B-Instruct	82.3 (+57.3)	37.0	39.0	39.0	37.0	38.0	28.0	48.0	41.0	28.0	37.2
	codegemma-1.1-7b-it	56.6 (+31.6)	5.0	1.0	4.0	6.0	6.0	1.0	3.0	3.0	3.0	3.6
	gemma-2-9b-it	89.2 (+64.2)	45.0	55.0	55.0	68.0	58.0	61.0	72.0	60.0	59.0	59.2
	Qwen2.5-Coder-7B-Instruct	68.1 (+43.1)	6.0	7.0	16.0	14.0	13.0	9.0	38.0	12.0	9.0	13.8
	Qwen2.5-7B-Instruct	81.4 (+56.4)	31.0	27.0	39.0	37.0	37.0	36.0	17.0	28.0	34.0	31.8
	Marco-o1	79.4 (+54.4)	31.0	26.0	36.0	36.0	27.0	38.0	40.0	28.0	28.0	32.2
	deepseek-coder-7b-instruct-v1.5	59.5 (+34.5)	7.0	4.0	9.0	4.0	8.0	5.0	9.0	2.0	3.0	5.7
	deepseek-llm-7b-chat	46.6 (+21.6)	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	4.0	0.6
	DeepSeek-R1-Distill-Qwen-7B	33.4 (+8.4)	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.2
	DeepSeek-R1-Distill-Llama-8B	30.2 (+5.2)	1.0	0.0	0.0	0.0	2.0	0.0	1.0	0.0	2.0	0.7
	Falcon3-7B-Instruct	76.9 (+51.9)	18.0	25.0	28.0	30.0	37.0	26.0	43.0	30.0	25.0	29.1
	Baichuan2-7B-Chat	40.6 (+15.6)	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.1
	Yi-Coder-9B-Chat	62.6 (+37.6)	2.0	3.0	9.0	10.0	8.0	4.0	8.0	8.0	1.0	5.9
	Yi-1.5-9B-Chat	85.7 (+60.7)	33.0	42.0	37.0	44.0	43.0	40.0	51.0	42.0	40.0	41.3
	granite-8b-code-instruct-128k	75.5 (+50.5)	12.0	13.0	19.0	19.0	24.0	14.0	18.0	16.0	9.0	16.0
	granite-3.0-8b-instruct	77.1 (+52.1)	20.0	15.0	28.0	35.0	27.0	28.0	41.0	29.0	17.0	26.7
	EXAONE-3.5-7.8B-Instruct	79.7 (+54.7)	28.0	21.0	32.0	48.0	38.0	33.0	40.0	32.0	29.0	33.4
≤16B	CodeLlama-13b-Instruct-hf	48.0 (+23.0)	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
	Qwen2.5-Coder-14B-Instruct	82.4 (+57.4)	42.0	31.0	49.0	49.0	49.0	50.0	61.0	47.0	42.0	46.7
	Qwen2.5-14B-Instruct	91.8 (+66.8)	60.0	59.0	64.0	72.0	60.0	74.0	74.0	60.0	66.0	65.4
	DeepSeek-Coder-V2-Lite-Instruct	64.1 (+39.1)	11.0	6.0	6.0	7.0	9.0	7.0	8.0	4.0	4.0	6.9
	DeepSeek-V2-Lite-Chat	62.5 (+37.5)	2.0	3.0	10.0	9.0	11.0	7.0	13.0	5.0	7.0	7.4
	DeepSeek-R1-Distill-Qwen-14B	88.3 (+63.3)	46.0	47.0	62.0	59.0	58.0	59.0	68.0	61.0	62.0	58.0
	falcon-11B	46.6 (+21.6)	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
	Falcon3-10B-Instruct	71.1 (+46.1)	9.0	16.0	18.0	17.0	21.0	10.0	27.0	14.0	8.0	15.6
	Baichuan2-13B-Chat	39.1 (+14.1)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	WizardLM-13B-V1.2	40.8 (+15.8)	1.0	0.0	0.0	0.0	1.0	0.0	2.0	0.0	0.0	0.4
	Phi-3-medium-128k-instruct	87.7 (+62.7)	41.0	42.0	52.0	50.0	52.0	56.0	62.0	54.0	52.0	51.2
	phi-4	87.7 (+62.7)	39.0	45.0	55.0	53.0	54.0	51.0	59.0	52.0	50.0	50.9
	starcoder2-15b-instruct-v0.1	39.6 (+14.6)	0.0	0.0	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.3
	Mistral-Nemo-Instruct-2407	78.7 (+53.7)	25.0	24.0	35.0	36.0	32.0	33.0	43.0	35.0	31.0	32.7
≤34B	CodeLlama-34b-Instruct-hf	46.1 (+21.1)	1.0	0.0	3.0	3.0	3.0	4.0	0.0	0.0	2.0	1.8
	gemma-2-27b-it	93.3 (+68.3)	64.0	62.0	**79.0**	71.0	68.0	72.0	76.0	72.0	67.0	70.1
	Qwen2.5-Coder-32B-Instruct	93.2 (+68.2)	64.0	63.0	74.0	74.0	74.0	76.0	76.0	71.0	**72.0**	71.6
	Qwen2.5-32B-Instruct	92.7 (+67.7)	58.0	58.0	76.0	**81.0**	72.0	77.0	77.0	70.0	69.0	70.9
	QwQ-32B	91.8 (+66.8)	59.0	57.0	70.0	70.0	63.0	72.0	78.0	66.0	65.0	66.7
	Sky-T1-32B-Preview	88.2 (+63.2)	47.0	49.0	66.0	64.0	56.0	66.0	70.0	58.0	58.0	59.3
	deepseek-coder-33b-instruct	49.2 (+24.2)	0.0	3.0	3.0	6.0	2.0	3.0	8.0	10.0	3.0	4.2
	DeepSeek-R1-Distill-Qwen-32B	91.0 (+66.0)	56.0	60.0	77.0	68.0	66.0	74.0	74.0	70.0	69.0	68.2
	Yi-1.5-34B-Chat	75.6 (+50.6)	25.0	28.0	30.0	27.0	29.0	29.0	30.0	27.0	27.0	28.0
	Mistral-Small-Instruct-2409	81.3 (+56.3)	33.0	33.0	40.0	33.0	37.0	37.0	59.0	36.0	38.0	38.4
	granite-34b-code-instruct-8k	74.0 (+49.0)	19.0	14.0	24.0	28.0	28.0	24.0	30.0	23.0	21.0	23.4
	internlm2_5-20b-chat	91.5 (+66.5)	59.0	63.0	66.0	67.0	64.0	61.0	75.0	65.0	50.0	63.3
	EXAONE-3.5-32B-Instruct	87.6 (+62.6)	43.0	50.0	47.0	53.0	51.0	48.0	65.0	51.0	48.0	50.7
≤72B	CodeLlama-70b-Instruct-hf	74.0 (+49.0)	8.0	9.0	14.0	10.0	13.0	17.0	31.0	16.0	11.0	14.3
	Llama-3.1-70B-Instruct	94.3 (+69.3)	**72.0**	**68.0**	74.0	**81.0**	73.0	**78.0**	**82.0**	**74.0**	70.0	**74.7**
	Llama-3.3-70B-Instruct	94.3 (+69.3)	65.0	**68.0**	76.0	80.0	**76.0**	**78.0**	**82.0**	73.0	**72.0**	74.4
	Qwen2.5-72B-Instruct	91.7 (+66.7)	46.0	53.0	71.0	74.0	64.0	69.0	73.0	63.0	65.0	64.2
	deepseek-llm-67b-chat	80.6 (+55.6)	36.0	28.0	47.0	44.0	49.0	38.0	60.0	45.0	35.0	42.4
	DeepSeek-R1-Distill-Llama-70B	83.5 (+58.5)	45.0	44.0	62.0	51.0	53.0	57.0	71.0	55.0	57.0	55.0
	WizardLM-70B-V1.0	75.2 (+50.2)	25.0	16.0	25.0	24.0	30.0	19.0	30.0	29.0	22.0	24.4
	K2-Chat	81.6 (+56.6)	39.0	31.0	45.0	45.0	43.0	37.0	59.0	50.0	31.0	42.2
	falcon-40b-instruct	37.4 (+12.4)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

PPA: Proportion of Plurarity Agreement, **Best Score Overall**, Best Score within Scale

Table 10: Change Localisation (Easy) - Proportion of Plurarity Agreement and Invariant Accuracy (%).

Scale	Model	PPA (Random = 25.0)	C	C++	CSharp	Go	Java	JavaScript	PHP	Python	Ruby	Overall
≤3B	Llama-3.2-3B-Instruct	45.8 (+20.8)	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	0.4
	Llama-3.2-1B-Instruct	31.3 (+6.3)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Qwen2.5-Coder-3B-Instruct	52.1 (+27.1)	0.0	3.0	0.0	2.0	2.0	1.0	3.0	2.0	3.0	1.8
	Qwen2.5-Coder-1.5B-Instruct	27.0 (+2.0)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Qwen2.5-3B-Instruct	78.0 (+53.0)	34.0	27.0	39.0	31.0	39.0	28.0	45.0	34.0	38.0	35.0
	Qwen2.5-1.5B-Instruct	54.5 (+29.5)	0.0	1.0	1.0	1.0	2.0	1.0	4.0	0.0	1.0	1.2
	deepseek-coder-1.3b-instruct	25.1 (+0.1)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	DeepSeek-R1-Distill-Qwen-1.5B	25.0 (+0.0)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Falcon3-3B-Instruct	54.5 (+29.5)	0.0	3.0	1.0	5.0	2.0	1.0	2.0	0.0	1.0	1.7
	Falcon3-1B-Instruct	25.1 (+0.1)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Phi-3-mini-128k-instruct	72.9 (+47.9)	21.0	21.0	29.0	36.0	26.0	27.0	48.0	36.0	32.0	30.7
	Yi-Coder-1.5B-Chat	36.2 (+11.2)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	granite-3b-code-instruct-128k	42.0 (+17.0)	0.0	1.0	0.0	1.0	0.0	1.0	2.0	1.0	0.0	0.7
	granite-3.0-3b-a800m-instruct	43.2 (+18.2)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	granite-3.0-2b-instruct	49.4 (+24.4)	2.0	2.0	1.0	0.0	0.0	1.0	3.0	0.0	0.0	1.0
	EXAONE-3.5-2.4B-Instruct	40.0 (+15.0)	1.0	0.0	1.0	0.0	0.0	3.0	1.0	1.0	0.0	0.8
	internlm2_5-1_8b-chat	48.4 (+23.4)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	stable-code-instruct-3b	39.1 (+14.1)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
≤9B	CodeLlama-7b-Instruct-hf	45.6 (+20.6)	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.3
	Llama-3.1-8B-Instruct	76.1 (+51.1)	27.0	30.0	29.0	27.0	29.0	28.0	41.0	32.0	25.0	29.8
	codegemma-1.1-7b-it	52.9 (+27.9)	3.0	1.0	4.0	5.0	5.0	1.0	3.0	3.0	3.0	3.1
	gemma-2-9b-it	85.7 (+60.7)	41.0	45.0	49.0	58.0	49.0	55.0	68.0	50.0	53.0	52.0
	Qwen2.5-Coder-7B-Instruct	65.7 (+40.7)	6.0	5.0	15.0	13.0	12.0	8.0	18.0	10.0	9.0	10.7
	Qwen2.5-7B-Instruct	79.9 (+54.9)	26.0	28.0	41.0	32.0	36.0	40.0	43.0	28.0	34.0	34.2
	Marco-o1	77.2 (+52.2)	25.0	26.0	30.0	27.0	31.0	34.0	40.0	24.0	30.0	29.7
	deepseek-coder-7b-instruct-v1.5	54.8 (+29.8)	5.0	4.0	6.0	2.0	5.0	5.0	9.0	2.0	3.0	4.6
	deepseek-llm-7b-chat	43.9 (+18.9)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.4
	DeepSeek-R1-Distill-Qwen-7B	32.4 (+7.4)	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.2
	DeepSeek-R1-Distill-Llama-8B	30.2 (+5.2)	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	2.0	0.6
	Falcon3-7B-Instruct	70.8 (+45.8)	10.0	20.0	21.0	21.0	26.0	20.0	38.0	25.0	22.0	22.6
	Baichuan2-7B-Chat	38.3 (+13.3)	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.1
	Yi-Coder-9B-Chat	58.4 (+33.4)	1.0	1.0	4.0	7.0	4.0	5.0	8.0	4.0	1.0	3.9
	Yi-1.5-9B-Chat	85.5 (+60.5)	36.0	46.0	40.0	44.0	45.0	43.0	54.0	45.0	36.0	43.2
	granite-8b-code-instruct-128k	71.1 (+46.1)	10.0	9.0	16.0	15.0	17.0	11.0	15.0	10.0	11.0	12.7
	granite-3.0-8b-instruct	67.6 (+42.6)	15.0	11.0	23.0	27.0	22.0	27.0	40.0	23.0	16.0	22.7
	EXAONE-3.5-7.8B-Instruct	73.2 (+48.2)	18.0	18.0	25.0	35.0	29.0	29.0	37.0	28.0	27.0	27.3
≤16B	CodeLlama-13b-Instruct-hf	44.9 (+19.9)	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1
	Qwen2.5-Coder-14B-Instruct	74.5 (+49.5)	29.0	22.0	37.0	38.0	37.0	45.0	56.0	36.0	36.0	37.3
	Qwen2.5-14B-Instruct	89.6 (+64.6)	51.0	48.0	58.0	59.0	55.0	67.0	69.0	46.0	61.0	57.1
	DeepSeek-Coder-V2-Lite-Instruct	61.3 (+36.3)	9.0	4.0	3.0	7.0	8.0	7.0	8.0	4.0	6.0	6.2
	DeepSeek-V2-Lite-Chat	59.5 (+34.5)	2.0	3.0	6.0	10.0	9.0	7.0	14.0	4.0	7.0	6.9
	DeepSeek-R1-Distill-Qwen-14B	85.8 (+60.8)	43.0	39.0	57.0	51.0	52.0	57.0	67.0	50.0	63.0	53.2
	falcon-11B	43.3 (+18.3)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Falcon3-10B-Instruct	66.3 (+41.3)	6.0	14.0	15.0	17.0	11.0	8.0	20.0	9.0	8.0	12.0
	Baichuan2-13B-Chat	37.1 (+12.1)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	WizardLM-13B-V1.2	38.4 (+13.4)	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.1
	Phi-3-medium-128k-instruct	82.0 (+57.0)	38.0	29.0	45.0	38.0	39.0	48.0	57.0	44.0	49.0	43.0
	phi-4	84.0 (+59.0)	35.0	36.0	49.0	42.0	45.0	49.0	59.0	42.0	46.0	44.8
	starcoder2-15b-instruct-v0.1	38.0 (+13.0)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Mistral-Nemo-Instruct-2407	78.0 (+53.0)	19.0	23.0	32.0	32.0	29.0	31.0	40.0	25.0	30.0	29.0
≤34B	CodeLlama-34b-Instruct-hf	45.8 (+20.8)	1.0	0.0	3.0	3.0	3.0	4.0	0.0	0.0	2.0	1.8
	gemma-2-27b-it	87.9 (+62.9)	52.0	48.0	67.0	52.0	57.0	63.0	67.0	61.0	61.0	58.7
	Qwen2.5-Coder-32B-Instruct	89.3 (+64.3)	53.0	55.0	66.0	66.0	64.0	74.0	76.0	56.0	**69.0**	64.3
	Qwen2.5-32B-Instruct	91.8 (+66.8)	58.0	58.0	**70.0**	**75.0**	66.0	74.0	77.0	61.0	**69.0**	67.6
	QwQ-32B	90.5 (+65.5)	53.0	52.0	67.0	66.0	64.0	**77.0**	75.0	59.0	61.0	63.8
	Sky-T1-32B-Preview	88.3 (+63.3)	41.0	48.0	60.0	63.0	53.0	68.0	72.0	49.0	57.0	56.8
	deepseek-coder-33b-instruct	43.9 (+18.9)	0.0	3.0	3.0	4.0	1.0	1.0	8.0	6.0	3.0	3.2
	DeepSeek-R1-Distill-Qwen-32B	89.9 (+64.9)	54.0	58.0	65.0	67.0	61.0	74.0	73.0	61.0	68.0	64.6
	Yi-1.5-34B-Chat	74.1 (+49.1)	19.0	22.0	28.0	27.0	20.0	31.0	30.0	21.0	23.0	24.6
	Mistral-Small-Instruct-2409	75.2 (+50.2)	21.0	24.0	33.0	29.0	29.0	32.0	51.0	32.0	33.0	31.6
	granite-34b-code-instruct-8k	70.1 (+45.1)	16.0	13.0	21.0	25.0	18.0	21.0	29.0	21.0	22.0	20.7
	internlm2_5-20b-chat	88.2 (+63.2)	48.0	50.0	60.0	56.0	56.0	59.0	68.0	53.0	48.0	55.3
	EXAONE-3.5-32B-Instruct	83.7 (+58.7)	36.0	39.0	48.0	49.0	47.0	49.0	61.0	46.0	47.0	46.9
≤72B	CodeLlama-70b-Instruct-hf	69.7 (+44.7)	5.0	6.0	11.0	9.0	8.0	16.0	27.0	9.0	11.0	11.3
	Llama-3.1-70B-Instruct	92.5 (+67.5)	**65.0**	**61.0**	67.0	73.0	**68.0**	76.0	**80.0**	**66.0**	65.0	**69.0**
	Llama-3.3-70B-Instruct	91.4 (+66.4)	57.0	58.0	65.0	71.0	65.0	**77.0**	77.0	61.0	68.0	66.6
	Qwen2.5-72B-Instruct	89.4 (+64.4)	42.0	47.0	60.0	62.0	61.0	65.0	71.0	50.0	67.0	58.3
	deepseek-llm-67b-chat	78.5 (+53.5)	28.0	26.0	40.0	40.0	41.0	36.0	58.0	40.0	31.0	37.8
	DeepSeek-R1-Distill-Llama-70B	77.8 (+52.8)	37.0	38.0	45.0	42.0	46.0	52.0	67.0	46.0	53.0	47.3
	WizardLM-70B-V1.0	68.3 (+43.3)	22.0	9.0	18.0	20.0	21.0	18.0	25.0	20.0	20.0	19.2
	K2-Chat	79.4 (+54.4)	29.0	29.0	40.0	47.0	38.0	37.0	56.0	47.0	32.0	39.4
	falcon-40b-instruct	38.0 (+13.0)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

PPA: Proportion of Plurarity Agreement, **Best Score Overall**, Best Score within Scale

Table 11: Change Localisation (Hard) - Proportion of Plurarity Agreement and Invariant Accuracy (%).

Scale	Model	PPA (Random = 25.0)	C	C++	CSharp	Go	Java	JavaScript	PHP	Python	Ruby	Overall
≤3B	Llama-3.2-3B-Instruct	49.3 (+24.3)	3.8	5.9	11.7	16.3	3.5	7.0	7.8	9.0	23.9	9.9
	Llama-3.2-1B-Instruct	31.5 (+6.5)	2.5	1.2	1.3	3.5	1.2	2.3	1.1	5.1	1.1	2.1
	Qwen2.5-Coder-3B-Instruct	44.5 (+19.5)	11.3	11.8	15.6	11.6	17.4	7.0	16.7	15.4	3.4	12.2
	Qwen2.5-Coder-1.5B-Instruct	42.7 (+17.7)	5.0	5.9	5.2	2.3	2.3	4.7	10.0	10.3	5.7	5.7
	Qwen2.5-3B-Instruct	82.5 (+57.5)	50.0	61.2	51.9	53.5	59.3	57.0	62.2	51.3	51.1	55.3
	Qwen2.5-1.5B-Instruct	85.5 (+60.5)	32.5	52.9	46.8	40.7	44.2	44.2	57.8	39.7	52.3	45.7
	deepseek-coder-1.3b-instruct	28.5 (+3.5)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.3	0.0	0.1
	DeepSeek-R1-Distill-Qwen-1.5B	25.0 (+0.0)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Falcon3-3B-Instruct	70.4 (+45.4)	15.0	30.6	27.3	29.0	29.1	29.1	37.8	32.1	21.6	27.9
	Falcon3-1B-Instruct	29.2 (+4.2)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Phi-3-mini-128k-instruct	86.3 (+61.3)	53.8	55.3	59.7	54.7	62.8	69.8	56.7	55.1	55.7	58.2
	Yi-Coder-1.5B-Chat	37.4 (+12.4)	0.0	1.2	0.0	1.2	0.0	0.0	0.0	0.0	0.0	0.3
	granite-3b-code-instruct-128k	41.8 (+16.8)	3.8	2.4	0.0	4.7	2.3	0.0	3.3	6.4	3.4	2.9
	granite-3.0-3b-a800m-instruct	67.0 (+42.0)	7.5	8.2	6.5	5.8	9.3	8.1	11.1	5.1	3.4	7.2
	granite-3.0-2b-instruct	67.0 (+42.0)	20.0	21.2	14.3	22.1	18.6	22.1	21.1	23.1	9.1	19.1
	EXAONE-3.5-2.4B-Instruct	76.8 (+51.8)	20.0	40.0	40.3	31.4	36.0	30.2	44.4	41.0	25.0	34.3
	internlm2_5-1_8b-chat	50.1 (+25.1)	1.3	0.0	0.0	8.1	5.8	5.8	3.3	5.1	1.1	3.4
	stable-code-instruct-3b	36.2 (+11.2)	1.3	0.0	2.6	1.2	0.0	3.5	2.2	2.6	0.0	1.5
≤9B	CodeLlama-7b-Instruct-hf	48.9 (+23.9)	3.8	2.4	3.9	3.5	4.7	7.0	11.1	12.8	3.4	5.8
	Llama-3.1-8B-Instruct	89.2 (+64.2)	57.5	64.7	70.1	67.4	67.4	68.6	68.9	67.9	58.0	65.6
	codegemma-1.1-7b-it	68.0 (+43.0)	20.0	23.5	20.8	17.4	19.8	20.9	16.7	24.4	14.8	19.8
	gemma-2-9b-it	87.1 (+62.1)	57.5	57.6	68.8	54.7	48.8	58.1	58.9	66.7	58.0	58.8
	Qwen2.5-Coder-7B-Instruct	89.9 (+64.9)	58.8	62.4	75.3	66.3	73.3	72.1	68.9	65.4	65.9	67.6
	Qwen2.5-7B-Instruct	89.0 (+64.0)	66.3	57.6	66.2	60.5	65.1	62.8	62.2	64.1	61.4	62.9
	Marco-o1	91.2 (+66.2)	62.5	64.7	70.1	66.3	75.6	70.9	73.3	67.9	72.7	69.3
	deepseek-coder-7b-instruct-v1.5	62.6 (+37.6)	15.0	10.6	11.7	20.9	15.1	14.0	18.9	20.5	12.5	15.5
	deepseek-llm-7b-chat	54.7 (+29.7)	7.5	8.2	9.1	12.8	12.8	12.8	7.8	15.4	9.1	10.6
	DeepSeek-R1-Distill-Qwen-7B	27.0 (+2.0)	0.0	0.0	0.0	1.2	0.0	0.0	0.0	0.0	0.0	0.1
	DeepSeek-R1-Distill-Llama-8B	53.8 (+28.8)	13.8	7.1	13.0	15.1	9.3	9.3	13.3	14.1	13.6	12.1
	Falcon3-7B-Instruct	76.7 (+51.7)	50.0	54.1	42.9	47.7	50.0	36.0	40.0	39.7	38.6	44.3
	Baichuan2-7B-Chat	40.5 (+15.5)	0.0	2.4	1.3	5.8	2.3	2.3	3.3	6.4	5.7	3.3
	Yi-Coder-9B-Chat	81.1 (+56.1)	40.0	43.5	50.6	39.5	33.7	39.5	43.3	44.9	33.0	40.9
	Yi-1.5-9B-Chat	89.8 (+64.8)	60.0	68.2	64.9	59.3	70.9	72.1	61.1	71.8	65.9	66.0
	granite-8b-code-instruct-128k	73.8 (+48.8)	15.0	25.9	15.6	24.4	16.3	19.8	23.3	30.8	29.5	22.3
	granite-3.0-8b-instruct	85.6 (+60.6)	40.0	47.1	44.2	43.0	40.7	41.9	50.0	29.5	29.5	40.6
	EXAONE-3.5-7.8B-Instruct	90.3 (+65.3)	60.0	69.4	71.4	65.1	68.6	72.1	76.7	66.7	65.9	68.4
≤16B	CodeLlama-13b-Instruct-hf	69.1 (+44.1)	2.5	8.2	13.0	11.6	11.6	15.1	14.4	23.1	25.0	13.8
	Qwen2.5-Coder-14B-Instruct	90.9 (+65.9)	62.5	55.3	72.7	66.3	75.6	61.6	67.8	61.5	65.9	65.5
	Qwen2.5-14B-Instruct	97.4 (+72.4)	90.0	83.5	90.9	89.5	88.4	88.4	87.8	83.3	88.6	87.8
	DeepSeek-Coder-V2-Lite-Instruct	58.8 (+33.8)	22.5	17.6	13.0	12.8	17.4	15.1	14.4	24.4	13.6	16.8
	DeepSeek-V2-Lite-Chat	69.8 (+44.8)	27.5	34.1	28.6	26.0	23.3	20.9	24.4	28.2	18.2	25.7
	DeepSeek-R1-Distill-Qwen-14B	94.0 (+69.0)	72.5	72.9	88.3	77.9	77.9	83.7	80.0	66.7	75.0	77.2
	falcon-11B	47.9 (+22.9)	8.8	10.6	9.1	10.5	2.3	10.5	10.0	15.4	5.7	9.2
	Falcon3-10B-Instruct	73.9 (+48.9)	42.5	32.9	23.4	18.6	23.3	17.4	30.0	11.5	14.8	23.8
	Baichuan2-13B-Chat	29.9 (+4.9)	0.0	0.0	1.3	0.0	0.0	2.3	1.1	5.1	0.0	1.1
	WizardLM-13B-V1.2	39.4 (+14.4)	6.3	5.9	2.6	11.6	1.2	9.3	5.6	9.0	2.3	6.0
	Phi-3-medium-128k-instruct	87.0 (+62.0)	57.5	48.2	61.0	58.1	62.8	59.3	64.4	60.3	65.9	59.7
	phi-4	96.3 (+71.3)	82.5	81.2	93.5	80.2	84.9	90.7	84.4	80.8	81.8	84.4
	starcoder2-15b-instruct-v0.1	63.8 (+38.8)	11.3	14.1	20.8	15.1	11.6	14.0	23.3	25.6	18.2	17.1
	Mistral-Nemo-Instruct-2407	88.0 (+63.0)	57.5	58.8	67.5	62.8	65.1	55.8	57.8	61.5	65.9	61.4
≤34B	CodeLlama-34b-Instruct-hf	71.5 (+46.5)	7.5	11.8	18.2	12.8	15.1	17.4	10.0	11.5	18.2	13.6
	gemma-2-27b-it	94.0 (+69.0)	80.0	78.8	79.2	72.1	83.7	72.1	72.2	70.5	77.3	76.2
	Qwen2.5-Coder-32B-Instruct	98.3 (+73.3)	91.3	84.7	96.1	94.2	91.9	96.5	94.4	85.9	89.8	91.6
	Qwen2.5-32B-Instruct	98.3 (+73.3)	90.0	91.8	96.1	89.5	89.5	91.9	96.7	89.7	**95.5**	92.3
	QwQ-32B	95.2 (+70.2)	75.0	77.6	93.5	80.2	86.0	83.7	85.6	73.1	87.5	82.5
	Sky-T1-32B-Preview	96.8 (+71.8)	80.0	84.7	93.5	82.6	86.0	89.5	94.4	80.8	92.0	87.1
	deepseek-coder-33b-instruct	51.2 (+26.2)	6.3	3.5	6.5	12.8	4.7	11.6	15.6	16.7	13.6	10.1
	DeepSeek-R1-Distill-Qwen-32B	93.6 (+68.6)	78.8	78.8	87.0	86.0	79.1	88.4	85.6	71.8	81.8	81.9
	Yi-1.5-34B-Chat	94.0 (+69.0)	66.3	72.9	72.7	62.8	70.9	77.9	78.9	70.5	70.5	71.5
	Mistral-Small-Instruct-2409	90.9 (+65.9)	47.5	64.7	71.4	61.6	60.5	69.8	76.7	62.8	59.1	63.8
	granite-34b-code-instruct-8k	76.7 (+51.7)	21.3	16.5	18.2	32.6	16.3	18.6	27.8	38.5	27.3	24.1
	internlm2_5-20b-chat	95.7 (+70.7)	83.8	80.0	89.6	80.2	82.6	81.4	85.6	76.9	79.5	82.2
	EXAONE-3.5-32B-Instruct	94.6 (+69.6)	76.3	82.4	94.8	76.7	70.9	73.3	76.7	78.2	79.5	78.8
≤72B	CodeLlama-70b-Instruct-hf	78.6 (+53.6)	32.5	38.8	42.9	37.2	34.9	46.5	42.2	43.6	38.6	39.7
	Llama-3.1-70B-Instruct	96.0 (+71.0)	85.0	77.6	88.3	83.7	87.2	90.7	86.7	74.4	84.1	84.2
	Llama-3.3-70B-Instruct	95.9 (+70.9)	86.3	76.5	89.6	83.7	83.7	90.7	87.8	75.6	87.5	84.6
	Qwen2.5-72B-Instruct	99.6 (+74.6)	**97.5**	**98.8**	**97.4**	**96.5**	**98.8**	**97.7**	**97.8**	**93.6**	**95.5**	**97.1**
	deepseek-llm-67b-chat	83.5 (+58.5)	55.0	49.4	50.6	58.1	50.0	57.0	55.6	55.1	59.1	54.4
	DeepSeek-R1-Distill-Llama-70B	94.3 (+69.3)	71.3	77.6	74.0	81.4	79.1	84.9	80.0	76.9	76.1	77.9
	WizardLM-70B-V1.0	87.3 (+62.3)	52.5	52.9	62.3	55.8	60.5	61.6	62.2	56.4	55.7	57.8
	K2-Chat	93.0 (+68.0)	73.8	64.7	90.9	72.1	79.1	72.1	77.8	75.6	64.8	74.5
	falcon-40b-instruct	47.2 (+22.2)	2.5	5.9	10.4	9.3	2.3	8.1	7.8	12.8	1.1	6.7

PPA: Proportion of Plurarity Agreement, **Best Score Overall**, Best Score within Scale

Table 12: Solution Identification (Easy) - Proportion of Plurarity Agreement and Invariant Accuracy (%).

Scale	Model	PPA (Random = 25.0)	C	C++	CSharp	Go	Java	JavaScript	PHP	Python	Ruby	Overall
≤3B	Llama-3.2-3B-Instruct	48.6 (+23.6)	5.0	4.7	7.8	16.3	1.2	7.0	6.7	7.7	12.5	7.6
	Llama-3.2-1B-Instruct	31.2 (+6.2)	0.0	1.2	0.0	1.2	0.0	0.0	0.0	3.8	1.1	0.8
	Qwen2.5-Coder-3B-Instruct	41.2 (+16.2)	5.0	4.7	6.5	11.6	11.6	11.6	10.0	9.0	2.2	8.0
	Qwen2.5-Coder-1.5B-Instruct	41.1 (+16.1)	3.8	1.2	1.3	4.7	1.2	0.0	3.3	6.4	4.6	2.9
	Qwen2.5-3B-Instruct	79.7 (+54.7)	51.3	50.6	36.4	47.7	48.8	45.4	45.6	41.0	39.8	45.2
	Qwen2.5-1.5B-Instruct	82.1 (+57.1)	28.8	45.9	39.0	29.1	48.8	34.9	43.3	26.9	43.2	37.8
	deepseek-coder-1.3b-instruct	27.5 (+2.5)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	DeepSeek-R1-Distill-Qwen-1.5B	25.1 (+0.1)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Falcon3-3B-Instruct	68.2 (+43.2)	10.0	23.5	15.6	20.9	26.7	19.8	31.1	15.4	12.5	19.5
	Falcon3-1B-Instruct	27.9 (+2.9)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Phi-3-mini-128k-instruct	83.1 (+58.1)	41.3	38.8	51.9	31.4	50.0	54.7	47.8	47.4	50.0	45.9
	Yi-Coder-1.5B-Chat	35.1 (+10.1)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	granite-3b-code-instruct-128k	41.1 (+16.1)	2.5	1.2	1.3	3.5	2.3	0.0	1.1	5.1	1.1	2.0
	granite-3.0-3b-a800m-instruct	62.3 (+37.3)	2.5	2.4	1.3	2.3	3.5	1.2	3.3	2.6	3.4	2.5
	granite-3.0-2b-instruct	64.3 (+39.3)	15.0	15.3	15.6	19.8	12.8	11.6	20.0	12.8	4.5	14.2
	EXAONE-3.5-2.4B-Instruct	73.2 (+48.2)	10.0	38.8	31.2	24.4	24.4	22.1	38.9	33.3	17.0	26.7
	internlm2_5-1_8b-chat	45.7 (+20.7)	0.0	1.2	1.3	5.8	2.3	1.2	2.2	2.6	0.0	1.8
	stable-code-instruct-3b	34.7 (+9.7)	1.3	1.2	1.3	2.3	1.2	0.0	4.4	2.6	0.0	1.6
≤9B	CodeLlama-7b-Instruct-hf	46.0 (+21.0)	3.8	2.4	1.3	3.5	2.3	2.3	7.8	5.1	2.3	3.4
	Llama-3.1-8B-Instruct	85.4 (+60.4)	45.0	55.3	63.6	55.8	52.3	57.0	57.8	53.8	47.7	54.3
	codegemma-1.1-7b-it	64.7 (+39.7)	15.0	17.6	13.0	15.1	18.6	15.1	17.8	14.1	9.1	15.0
	gemma-2-9b-it	85.0 (+60.0)	48.8	49.4	55.8	44.2	44.2	45.3	48.9	52.6	56.8	49.6
	Qwen2.5-Coder-7B-Instruct	86.8 (+61.8)	48.8	57.6	62.3	54.7	62.8	50.0	51.1	53.8	55.7	55.2
	Qwen2.5-7B-Instruct	85.6 (+60.6)	51.3	45.9	55.8	47.7	54.7	52.3	51.1	52.6	53.4	51.6
	Marco-o1	87.9 (+62.9)	60.0	51.8	67.5	53.5	62.8	55.8	52.2	62.8	56.8	58.1
	deepseek-coder-7b-instruct-v1.5	62.3 (+37.3)	6.3	8.2	9.1	16.3	11.6	12.8	7.8	17.9	12.5	11.4
	deepseek-llm-7b-chat	51.5 (+26.5)	0.0	2.4	3.9	7.0	7.0	5.8	5.6	11.5	4.5	5.3
	DeepSeek-R1-Distill-Qwen-7B	26.5 (+1.5)	0.0	0.0	0.0	1.2	0.0	0.0	0.0	0.0	0.0	0.1
	DeepSeek-R1-Distill-Llama-8B	49.8 (+24.8)	5.0	4.7	5.2	8.1	5.8	3.5	10.0	9.0	10.2	6.8
	Falcon3-7B-Instruct	73.5 (+48.5)	42.5	41.2	27.3	41.9	37.2	29.1	34.4	24.4	39.8	35.3
	Baichuan2-7B-Chat	37.7 (+12.7)	0.0	1.2	0.0	4.7	0.0	1.2	3.3	1.3	2.3	1.5
	Yi-Coder-9B-Chat	78.5 (+53.5)	32.5	40.0	33.8	37.2	26.7	31.4	27.8	37.2	30.7	33.0
	Yi-1.5-9B-Chat	87.9 (+62.9)	62.5	55.3	59.7	55.8	55.8	66.3	57.8	56.4	60.2	58.9
	granite-8b-code-instruct-128k	72.5 (+47.5)	8.8	12.9	15.6	14.0	9.3	19.8	20.0	26.9	14.8	15.8
	granite-3.0-8b-instruct	80.6 (+55.6)	20.0	25.9	28.6	31.4	31.4	19.8	26.7	19.2	25.0	25.3
	EXAONE-3.5-7.8B-Instruct	87.1 (+62.1)	57.5	54.1	59.7	55.8	59.3	58.1	57.8	52.6	58.0	57.0
≤16B	CodeLlama-13b-Instruct-hf	65.9 (+40.9)	7.5	7.1	9.1	10.5	5.8	11.6	13.3	10.3	14.8	10.0
	Qwen2.5-Coder-14B-Instruct	88.1 (+63.1)	55.0	45.9	59.7	51.2	65.1	55.8	56.7	53.8	62.5	56.2
	Qwen2.5-14B-Instruct	96.3 (+71.3)	83.8	84.7	83.1	80.2	82.6	84.9	81.1	73.1	84.1	81.9
	DeepSeek-Coder-V2-Lite-Instruct	57.4 (+32.4)	15.0	10.6	5.2	15.1	15.1	15.1	10.0	19.2	10.2	12.8
	DeepSeek-V2-Lite-Chat	66.9 (+41.9)	28.8	23.5	15.6	14.0	23.3	18.6	20.0	21.8	19.3	20.5
	DeepSeek-R1-Distill-Qwen-14B	91.5 (+66.5)	66.3	63.5	76.6	65.1	67.4	70.9	73.3	45.0	72.7	66.8
	falcon-11B	45.4 (+20.4)	12.5	5.9	3.9	8.1	3.5	9.3	8.9	14.1	6.8	8.1
	Falcon3-10B-Instruct	69.5 (+44.5)	30.0	23.5	18.2	15.1	25.6	7.0	20.0	11.5	11.4	18.0
	Baichuan2-13B-Chat	30.3 (+5.3)	0.0	0.0	1.3	1.2	0.0	1.2	1.1	1.3	0.0	0.7
	WizardLM-13B-V1.2	38.3 (+13.3)	6.3	3.5	0.0	10.5	1.2	5.8	7.8	3.8	2.3	4.6
	Phi-3-medium-128k-instruct	84.1 (+59.1)	53.8	41.2	49.4	47.7	57.0	50.0	55.6	47.4	55.7	50.8
	phi-4	94.5 (+69.5)	75.0	77.6	85.7	73.3	81.4	80.2	75.6	75.6	72.7	77.5
	starcoder2-15b-instruct-v0.1	64.5 (+39.5)	15.0	12.9	20.8	14.0	8.1	15.1	17.8	20.5	19.3	15.9
	Mistral-Nemo-Instruct-2407	85.2 (+60.2)	53.8	52.9	55.8	50.0	58.1	48.8	48.9	48.7	58.0	52.8
≤34B	CodeLlama-34b-Instruct-hf	70.8 (+45.8)	6.3	10.6	14.3	10.5	8.1	12.8	5.6	14.1	15.9	10.9
	gemma-2-27b-it	91.4 (+66.4)	67.5	68.2	64.9	67.4	69.8	64.0	67.8	61.5	60.2	65.7
	Qwen2.5-Coder-32B-Instruct	97.0 (+72.0)	87.5	80.0	90.9	89.5	**93.0**	86.0	82.2	78.2	84.1	85.7
	Qwen2.5-32B-Instruct	97.2 (+72.2)	90.0	89.4	**96.1**	**90.7**	86.0	86.0	**92.2**	85.9	88.6	89.5
	QwQ-32B	93.7 (+68.7)	68.8	71.8	89.6	75.6	76.7	76.7	82.2	70.5	83.0	77.2
	Sky-T1-32B-Preview	95.3 (+70.3)	78.8	82.4	90.9	76.7	80.2	81.4	85.6	76.9	84.1	81.9
	deepseek-coder-33b-instruct	50.3 (+25.3)	5.0	4.7	2.6	12.8	3.5	8.1	13.3	19.2	11.4	9.0
	DeepSeek-R1-Distill-Qwen-32B	92.2 (+67.2)	76.3	71.8	83.1	75.6	74.4	79.1	82.2	67.9	77.3	76.4
	Yi-1.5-34B-Chat	90.6 (+65.6)	56.3	61.2	48.1	52.3	62.8	52.3	61.1	57.7	61.4	57.0
	Mistral-Small-Instruct-2409	89.4 (+64.4)	42.5	64.7	66.2	57.0	65.1	61.6	67.8	60.3	55.7	60.1
	granite-34b-code-instruct-8k	75.8 (+50.8)	15.0	16.5	13.0	23.3	18.6	12.8	25.6	29.5	19.3	19.3
	internlm2_5-20b-chat	93.1 (+68.1)	75.0	71.8	72.7	64.0	74.4	74.4	73.3	62.8	68.2	70.7
	EXAONE-3.5-32B-Instruct	91.5 (+66.5)	76.3	67.1	80.5	60.5	67.4	67.4	70.0	65.4	70.5	69.4
≤72B	CodeLlama-70b-Instruct-hf	76.3 (+51.3)	22.5	35.3	35.1	26.7	33.7	41.9	30.0	32.1	37.5	32.7
	Llama-3.1-70B-Instruct	93.5 (+68.5)	83.8	67.1	75.3	74.4	80.2	81.4	81.1	65.4	81.8	76.7
	Llama-3.3-70B-Instruct	94.0 (+69.0)	78.8	71.8	80.5	79.1	75.6	81.4	84.4	67.9	83.0	78.0
	Qwen2.5-72B-Instruct	98.5 (+73.5)	**92.5**	**94.1**	89.6	89.5	90.7	**93.0**	88.9	**89.7**	**89.8**	**90.9**
	deepseek-llm-67b-chat	79.5 (+54.5)	48.8	34.1	41.6	52.3	46.5	50.0	54.4	48.7	46.6	47.0
	DeepSeek-R1-Distill-Llama-70B	91.7 (+66.7)	58.8	64.7	70.1	67.4	73.3	75.6	74.4	64.1	70.5	68.8
	WizardLM-70B-V1.0	83.5 (+58.5)	53.8	41.2	48.1	50.0	53.5	55.8	47.8	50.0	40.9	49.0
	K2-Chat	88.3 (+63.3)	63.8	58.8	74.0	61.6	64.0	60.5	64.4	60.3	47.7	61.7
	falcon-40b-instruct	45.4 (+20.4)	1.3	4.7	5.2	5.8	2.3	2.3	7.8	6.4	2.3	4.2

PPA: Proportion of Plurarity Agreement, **Best Score Overall**, Best Score within Scale

Table 13: Solution Identification (Hard) - Proportion of Plurarity Agreement and Invariant Accuracy (%).